Universität Bayreuth
Fakultät für Mathematik und Physik

Computergestützte Konstruktion von linearen Codes mit hoher Minimaldistanz unter Verwendung heuristischer Methoden

Diplomarbeit im Fach Mathematik

Eingereicht im Februar 2007 von

Johannes Zwanzger
Rotkreuzstraße 39
95447 Bayreuth
johannes.zwanzger@t-online.de

Inhaltsverzeichnis

Liste der Algorithmen

Einleitung

Das Anwendungsspektrum der Codierungstheorie ist sehr breit gefächert: Sei es die selbständige empfängerseitige Korrektur von Übertragungsfehlern im Rahmen elektronischer Kommunikation, die Fähigkeit eines CD-Players, auch verkratzte Scheiben noch einigermaßen wohlklingend wiedergeben zu können oder die Kontrolle der Korrektheit von Daten im Speicher eines Computers - in vielen, alltäglich gewordenen Gegenständen sind codierungstheoretische Verfahren im Einsatz.

Eine große Rolle spielen dabei, u.a. aufgrund Ihres hohen Strukturgrades, der leichten Beschreibbarkeit durch Generatormatrizen sowie einer relativ schnell möglichen Codierung bzw. Decodierung, die *linearen Codes*. Die Korrekturqualität eines Codes ist direkt gekoppelt mit der sogenannten *Minimaldistanz* des Codes: Je höher die Minimaldistanz, desto mehr Fehler können erkannt bzw. korrigiert werden. Im linearen Fall ist sie identisch mit dem *Minimalgewicht*, das ist die Mindestzahl an Nichtnullkomponenten, die jedes von Null verschiedene Wort des Codes besitzt. Eine Zielstellung aus der Theorie linearer Codes ist daher, zu gewissen vorgegebenen Parametern dieses Minimalgewicht zu maximieren.

Nur für kleine Parameter ist das durch vollständige Enumeration aller möglichen Codes erreichbar. Normalerweise wird deshalb meist wie folgt verfahren: Aus allgemeinen Überlegungen werden zu den vorgegebenen Parametern obere Schranken für die Minimaldistanz berechnet und dann nach Codes gesucht, die dieser Schranke möglichst nahe kommen - im Idealfall ist die Minimaldistanz gleich dem Schrankenwert. Häufig werden dabei Zusatzbedingungen an die Struktur der Codes gestellt, was sie leichter untersuchbar und eventuell auch unter kombinatorischen Gesichtspunkten interessanter macht; oft lassen sich die wesentlichen Eigenschaften dieser Codes sogar noch mit „Papier und Bleistift“, also ohne Computereinsatz, bestimmen. Nachteil solcher Vorgaben ist, dass Codes mit diesen Merkmalen oft nur für bestimmte Parameter existieren oder keine besonders hohe Minimaldistanz besitzen. Der in dieser Arbeit gewählte Ansatz geht in eine etwas andere Richtung: Er ist prinzipiell für beliebige Parameter einsetzbar und ausschließlich bei Computerunterstützung praktikabel. Kernstück ist eine heuristische Vergleichs- bzw. Bewertungsfunktion, die die Chance abschätzt, dass sich eine Matrix durch Anfügen von Spalten zur Generatormatrix eines Codes von vorgegebenen Länge und Minimaldistanz erweitern lässt. Ausgehend von einer beliebigen Startmatrix werden dann, grob gesprochen, sukzessive Spalten so angefügt, dass die Bewertung in jedem Schritt möglichst hoch bleibt. Der Raum, über den tatsächlich gesucht wird, bleibt so-

mit auf eine relativ kleine, „lokale Umgebung“ der im aktuellen Schritt vorliegenden Generatormatrix beschränkt.

Im ersten Kapitel werden noch einmal die wichtigsten Grundlagen der Codierungstheorie, soweit sie in dieser Arbeit benötigt werden, dargelegt und notationelle Fragen geklärt. Der zweite Abschnitt enthält grundsätzliche, das Verfahren motivierende Überlegungen sowie einen sehr simplen Entwurf für eine mögliche Suchroutine; das Vorhandensein eines Vergleichsoperators „>“ für Matrizen hinsichtlich ihrer Chance, zu einem Code mit den gewünschten Parametern erweiterbar zu sein, wird hier vorausgesetzt. Der konkreten Implementierung dieses Operators widmet sich Kapitel drei. Es werden zwei mögliche Varianten vorgestellt, die beide in meinem Programm ***„Heurico“*** implementiert und erfolgreich eingesetzt wurden. Das letzte Kapitel beschäftigt sich dann mit Modifikationen des Basisalgorithmus, vor allem mit der Vertiefung der Suche und dem Einsatz von Selektivität. Auch auf die Bedeutung (semi-)linearer Isometrie zwischen Codes für den Algorithmus wird eingegangen und am Ende ein heuristisches Kriterium zur schnellen Erkennung einer solchen vorgestellt. Trotz der sehr einfachen Arbeitsweise sind die Ergebnisse des Verfahrens durchaus vielversprechend: An 35 Stellen konnte ich bisher mit ***„Heurico“*** die internationalen Tabellen verbessern, zwei der neu gefundenen Codes sind sogar nachweisbar optimal hinsichtlich der Minimaldistanz. Eine genaue Auflistung findet sich in Anhang C, der Quellcode von ***„Heurico“*** auf der angefügten CD.

Alle Algorithmen sind der Übersichtlichkeit halber in Pseudocode geschrieben. Außerdem wurde der leichten Formulierbarkeit der Vorzug gegenüber einer (hinsichtlich des Programmablaufs) schnellen Implementierung gegeben; so lässt sich beispielsweise an vielen Stellen ausnutzen, dass die im Programm auftretenden Generatormatrizen auseinander durch Hinzufügen von Spalten hervorgehen und somit bereits für die Untermatrix berechnete Dinge, etwa die Gewichtsverteilung des generierten Codes, durch Updateformeln auch für die neue Matrix wiederverwendet werden können. Die Durchlaufreihenfolge durch Mengen wird in der Regel nicht konkret festgelegt („***foreach ... do***“), obwohl sie gelegentlich durchaus den Ablauf oder gar die Resultate beeinflussen kann. Auf diesen Einfluß wird jeweils bei der Beschreibung der Algorithmen hingewiesen; er ist stets „zufälliger“ Natur, d. h. für manche Beispiele mag zwar eine bestimmte Reihenfolge bessere Resultate liefern als eine andere, im Mittel, über viele Testläufe gesehen, werden sie sich aber als qualitativ gleichwertig herausstellen (sofern jeweils kein Zusatzwissen in die Sortierung eingeht). Die als vorhanden vorausgesetzten Hilfsfunktionen betreffen ausschließlich Routineaufgaben; ihre Implementierung findet sich in den meisten Fällen z. B. in der *C++*-Standardbibliothek.

An dieser Stelle sei allen Personen gedankt, die zur Entstehung dieser Diplomarbeit beigetragen haben, insbesondere Prof. Dr. Adalbert Kerber für seine Anregungen und die Motivation, aus dem ursprünglich als „kleine Spielerei“ gedachten Programmierprojekt eine Diplomarbeit zu machen. Herzlich bedanken möchte ich auch bei meinen lieben Eltern für ihre in jeder Hinsicht entgegengebrachte Unterstützung, die es mir ermöglichte, mich ganz auf mein Studium zu konzentrieren.

Kapitel 1

Grundlagen

In diesem einführenden Kapitel sollen kurz die wesentlichen Ideen und Grundbegriffe der Codierungstheorie behandelt werden, soweit sie für das Verständnis dieser Arbeit erforderlich sind. Es wird davon ausgegangen, dass der Leser die (wenigen) notwendigen algebraischen Vorkenntnisse mitbringt, also beispielsweise mit Matrizen und Strukturen wie Körpern, Vektorräumen, etc. vertraut ist.

1.1 Einsatz der Codierungstheorie in der Praxis

Eine praktische Anwendung der Codierungstheorie liegt in der Möglichkeit, die Kommunikation zwischen einem Sender und einem Empfänger über einen störanfälligen Kanal zu verbessern. Genauer gesagt erlaubt sie es dem Empfänger, zu erkennen, ob in einer empfangenen Nachricht Abweichungen gegenüber dem ursprünglich gesendeten Text vorliegen und diese ggf. ohne Rückfrage an den Sender zu korrigieren; beides funktioniert freilich nur bis zu einer gewissen Obergrenze an Übertragungsfehlern zuverlässig. Zunächst soll diese Situation mathematisch formuliert werden:

1.2 Mathematische Modellierung der Nachrichten

Grundsätzlich wollen wir davon ausgehen, dass sich Sender und Empfänger vor der Übertragung in allen Details über deren Ablauf abgesprochen haben. Weiterhin wird angenommen, dass in den Nachrichten stets nur Zeichen aus einem zuvor vereinbarten, endlichen Alphabet $\Sigma = \{\sigma_1, \sigma_2, ..., \sigma_q\}$ verwendet werden und jede Nachricht v ein Wort der Länge k über Σ ist. Die natürliche Zahl k ist die sog. *Nachrichtenlänge*. Die Menge V aller zulässigen Nachrichten ist dann $V = \Sigma^k$ und $|V| = q^k$. Es ist leicht einzusehen, dass obige Annahmen keine wesentliche Einschränkung gegenüber einer „gewöhnlichen“

Kommunikation, wie z.B. einer Unterhaltung zwischen zwei Personen, darstellen; schließlich kann jede Nachricht beliebiger Länge über irgendeinem Alphabet „künstlich“ in (ggf. mehrere) Nachrichten der Länge k über Σ übersetzt werden.

1.3 Notation und Begriffsklärung

Bevor es los geht, noch ein paar Vereinbarungen: Wir schreiben ein Wort u über Σ als Zeilenvektor, also $u = (u_1, u_2, ..., u_n)$ für ein Wort der Länge n. Das Zeichen u_i an der i-ten Stelle von u heißt auch die *i-te Koordinate von u*. Unter einer *einfachen Modifikation von u* wird die Abänderung einer einzelnen Koordinate von u verstanden, also eine Operation, durch die $u = (u_1, u_2, ..., u_n)$ in $u^* = (u_1, u_2, ..., u_{i-1}, u_i^*, u_{i+1}, ..., u_n)$ übergeht für ein i mit $1 \leq i \leq n$ und $u_i^* \in \Sigma \setminus \{u_i\}$. Entsprechend sei eine *t-fache Modifikation* die t-fache Hintereinanderausführung einfacher Modifikationen, wobei die Koordinaten der einzelnen Modifikationen paarweise verschiedenen sein müssen. Zwei Vektoren u, v mit $u = (u_1, ..., u_n)$ und $v = (v_1, ..., v_n)$ haben *Hammingabstand* (oder *Hammingdistanz*) t, wenn sie sich an genau t Koordinaten unterscheiden, also wenn $|\{i : u_i \neq v_i, 1 \leq i \leq n\}| = t$. Dies wird wie folgt notiert: $dist(u, v) = t$. Wir sagen, bei der Übertragung eines Vektors u sind t Fehler aufgetreten, wenn der empfangene Vektor u^* von u Hammingabstand t besitzt.

1.4 Grundprinzip der Fehlererkennung bzw. -korrektur

Anhand eines Beispiels soll nun erläutert werden, wie es dem Empfänger möglich wird, Fehler in einer empfangenen Nachricht zu erkennen und zu korrigieren. Es geht im Folgenden stets nur um die Übertragung einer einzelnen Nachricht $v = (v_1, v_2, ..., v_k)$. Betrachten wir den einfachen Fall $\Sigma = \{0, 1\}$ und $k = 3$. Die Nachrichtenmenge V enthält hier nur acht Elemente, nämlich:

$$V = \{(000), (001), (010), (011), (100), (101), (110), (111)\}$$

Nehmen wir zunächst an, alle Nachrichten aus V würden unverändert gesendet und es soll z. B. (101) verschickt werden. Wird durch einen Übertragungsfehler die „0“ in eine „1“ verfälscht, so wird (111) empfangen. Der Empfänger hat keine Möglichkeit zu erkennen, dass hier etwas schiefgelaufen ist, denn es könnte ja tatsächlich (111) gemeint gewesen sein. Allgemein gilt natürlich für alle Nachrichten $v \in V$, dass ein daraus durch eine beliebige Modifikation entstehendes Wort v^* wieder in V liegt; schließlich besteht V ja gerade aus *allen* Worten der Länge k über Σ. Bei unveränderter Übermittlung der

Nachrichten aus V kann also nicht ein einziger Fehler erkannt, geschweige denn korrigiert werden.
Um dies zu ermöglichen, müssen wir die Nachrichten vor dem Versenden „aufbereiten“: Wir ordnen jedem $v \in V$ durch eine (gleich noch genauer spezifizierte) Funktion γ injektiv ein Wort $\gamma(v)$ zu, das anstelle von v versandt wird; Ziel ist dabei, dass die neuen Nachrichten einen paarweise möglichst hohen Hammingabstand haben. Dann müssen schon relativ viele Fehler auftreten, damit eine von ihnen bei der Übertragung in eine andere verfälscht werden kann. Da es offensichtlich praktisch ist, wenn alle diese Ersatznachrichten ebenfalls Worte gleicher Länge über Σ sind und außerdem aufgrund der Injektivität mindestens q^k paarweise verschiedene solche Worte existieren müssen, ergibt sich, dass γ vom folgenden Typ sein sollte:

$$\gamma : V \rightarrowtail \Sigma^n \quad mit\, n \geq k.$$

Für $n = k$ haben wir allerdings nichts gewonnen, denn γ kann dann nur eine Permutation der Elemente aus V sein; interessant ist demnach nur der Fall $n > k$. Das bedeutet aber, dass alle Nachrichten aus V vor dem Versenden verlängert werden müssen. Die Möglichkeit, Fehler erkennen bzw. korrigieren zu können, wird also mit einer gewissen Steigerung des Übertragungsaufwands erkauft.
Mit C sei ab jetzt das Bild von V unter γ gemeint; dann ist $C \subset \Sigma^n$ und γ eine Bijektion zwischen V und C. Um nun, unabhängig von der Nachricht $v \in V$, mindestens t Übertragungsfehler erkennen zu können, muss sichergestellt sein, dass in C kein Wortpaar $c_1 \neq c_2$ existiert, für das $dist(c_1, c_2) \leq t$ gilt. Oder umgekehrt: Für die Größe

$$dist(C) := min\{dist(c_1, c_2) : c_1, c_2 \in C, c_1 \neq c_2\},$$

die wir als *Minimaldistanz von C* bezeichnen wollen, muss gelten: $dist(C) > t$.
Kommen wir zur Verdeutlichung wieder auf das Beispiel zurück. γ sei wie folgt definiert:

$$\gamma : \{0,1\}^3 \rightarrowtail \{0,1\}^6,$$

$$\begin{array}{lcl} (000) & \mapsto & (000000) \\ (001) & \mapsto & (001111) \\ (010) & \mapsto & (010110) \\ (011) & \mapsto & (011001) \\ (100) & \mapsto & (100101) \\ (101) & \mapsto & (101010) \\ (110) & \mapsto & (110011) \\ (111) & \mapsto & (111100) \end{array}$$

$C = \{(000000), (001111), (010110), (011001), (100101), (101010), (110011), (111100)\}$

Der Leser überzeuge sich bei Bedarf selbst durch Nachrechnen, dass hier $dist(C) = 3$ gilt. Es können also bis zu $t = 2$ Übertragungsfehler sicher erkannt werden. Möchten wir nun z. B. wieder (101) übermitteln, so senden wir stattdessen $\gamma((101)) = (101010)$; tritt jetzt dabei genau ein Fehler, etwa an der vierten Koordinate, auf, so kommt (101110) beim

Empfänger an. Der wird feststellen, dass dieses Wort nicht in C liegt und weiß somit, dass Fehler passiert sein müssen. Er könnte nun den Sender auffordern, die Nachricht doch bitte noch einmal zu schicken. Das kann aber, je nach Art der Übertragung, aufwändig oder gar unmöglich sein - besser wäre es, wenn er die ursprüngliche Nachricht selbst rekonstruieren kann. Ein naheliegendes Prinzip zur Fehlerkorrektur ist das folgende:

„Falls ein Wort $\tilde{c}$ empfangen wird, das nicht in C liegt, so bestimme stattdessen ein $\bar{c} \in C$, welches *möglichst nahe* an $\tilde{c}$ liegt, und gehe davon aus, dass ursprünglich $\bar{c}$ gesendet wurde".

Oder etwas mathematischer: Zu empfangenem $\tilde{c} \in \Sigma^n$ wähle $\bar{c} \in C$ so, dass
$dist(\tilde{c}, \bar{c}) = min\{dist(\tilde{c}, c) : c \in C\}$.

Die Auswahl von $\bar{c}$ aufgrund des Empfangs von $\tilde{c}$ heißt auch *Decodierung von $\tilde{c}$ in $\bar{c}$* und das beschriebene Verfahren ist unter dem Namen *Maximum-Likelihood-Decoding* (kurz *MLD*) bekannt. Voraussetzung für eine erfolgreiche Fehlerkorrektur ist natürlich, dass bei der Übertragung nicht zu viele Fehler passiert sind. Andernfalls ist die gesendete Nachricht derart verfälscht, dass ein anderes Wort aus C näher an dem empfangenen Vektor liegt als die Originalbotschaft. Es soll jetzt gezeigt werden, dass bei Verwendung von MLD und $dist(C) = t$ bis zu $\lfloor \frac{t-1}{2} \rfloor$ Fehler sicher korrigiert werden können:

Zunächst stellen wir fest, dass die bereits eingeführte Funktion dist: $\Sigma^n \times \Sigma^n \to \mathbb{N}$ eine Metrik ist, d.h. folgende Eigenschaften besitzt:

1. $\forall u, v \in \Sigma^n : dist(u, v) \geq 0 \quad und \quad [dist(u, v) = 0 \iff u = v]$ *(Nichtnegativität)*
2. $\forall u, v \in \Sigma^n : dist(u, v) = dist(v, u)$ *(Symmetrie)*
3. $\forall u, v, w \in \Sigma^n : dist(u, w) \leq dist(u, v) + dist(v, w)$ *(Dreiecksungleichung)*

□

Der Beweis ist sehr einfach und bleibt dem Leser überlassen. Der entscheidende Punkt ist die Dreiecksungleichung. Damit können wir den gewünschten Satz beweisen:

Satz 1.1 *Sei $C \subset \Sigma^n$ mit $dist(C) = t$, sowie $\bar{c} \in C$ (die ursprünglich gesendete Nachricht) und $\tilde{c} \in \Sigma^n$ (der empfangene Vektor). Es gelte $dist(\bar{c}, \tilde{c}) \leq \lfloor \frac{t-1}{2} \rfloor$. Dann folgt:*

$$\forall c \in C \setminus \{\bar{c}\} : dist(c, \tilde{c}) > \lfloor \frac{t-1}{2} \rfloor.$$

Das bedeutet, $\bar{c}$ hat unter allen Worten in C den kleinsten Abstand von $\tilde{c}$ und folglich wird $\tilde{c}$ durch MLD korrekt in $\bar{c}$ decodiert.

Beweis:
Angenommen, $\exists\, c \in C \setminus \{\bar{c}\}$ mit $dist(c, \tilde{c}) \leq \lfloor \frac{t-1}{2} \rfloor$. Dann folgt aus der Dreicksungleichung

und der Symmetrie von *dist*:

$$dist(\overline{c}, c) \leq dist(\overline{c}, \tilde{c}) + dist(\tilde{c}, c) \leq \lfloor \frac{t-1}{2} \rfloor + \lfloor \frac{t-1}{2} \rfloor \leq t-1$$

und damit wäre $dist(C) \leq t-1$, ein Widerspruch.

□

Greifen wir mit dieser Erkenntnis noch einmal das Beispiel auf: Da dort $dist(C) = 3$ war und $\lfloor \frac{3-1}{2} \rfloor = 1$, läßt sich ein einzelner Übertragungsfehler zuverlässig korrigieren. (101110) liegt in Hammingdistanz eins von (101010), folglich müsste es durch MLD korrekt in (101010) decodiert werden. Und in der Tat, wie man leicht nachprüft, haben alle anderen Worte aus C zu (101110) einen Hammingabstand von mindestens zwei; (101010) liegt demnach (101110) eindeutig am nächsten.

Zusammenfassend kann man sagen: Um zu vorgegebenem Σ, k und n mit $|\Sigma| = q$ eine möglichst gute Fehlerkorrektur zu erreichen, muss ein $\overline{C} \subset \Sigma^n$ mit $|\overline{C}| = q^k$ bestimmt werden, so dass $dist(\overline{C}) = max\{dist(C) : C \subset \Sigma^n, |C| = q^k\}$. Das genaue Aussehen der Bijektion γ spielt hingegen für die Korrekturqualität keine Rolle.

1.5 Lineare Codes

Bisher hatten wir zu gegebenem k und n keine weiteren Anforderungen an Σ und $C \subset \Sigma^n$ gestellt, abgesehen von $|C| = q^k$. Ein großer Teil der Codierungstheorie und auch die vorliegende Arbeit beschäftigt sich aber mit den sogenannten „linearen Codes“:

Σ wird dabei als endlicher Körper vorausgesetzt, also $\Sigma = \mathbb{F}_q$ (q Primzahlpotenz); der Nachrichtenraum ist $V = \mathbb{F}_q^k$. Als *linearen Code* bezeichnet man einen k-dimensionalen, linearen Unterraum $C \leq \mathbb{F}_q^n$; die Vektoren aus C heißen die *Codeworte* und n die *Blocklänge* von C. Für ein Codewort $c \in C$ nennen wir $dist(c, \vec{0})$, also den Hammingabstand zwischen c und dem Nullvektor $\vec{0} \in \mathbb{F}_q^n$, das *Gewicht von c*, kurz $gew(c)$. Offensichtlich ist dieses genau die Anzahl der Nichtnull-Komponenten von c. Hat ein k-dimensionaler linearer Code C der Blocklänge n über $\mathbb{F}_q$ die Minimaldistanz d, so sagt man auch, C habe die Parameter (n, k, d, q) bzw. C sei ein linearer (n, k, d, q)-Code.

Welchen Vorteil hat diese auf den ersten Blick vielleicht künstlich erscheinende Einschränkung? Nun, lineare Codes sind aufgrund der in großem Maße vorhandenen Struktur an vielen Stellen leichter zu handhaben als strukturlose Mengen von Worten über einem beliebigen Alphabet:

Zunächst einmal läßt sich die Minimaldistanz $dist(C)$ viel einfacher bestimmen. Da es sich bei C um einen Vektorraum handelt, ist, für zwei beliebige Codeworte $c_1 \neq c_2 \in C$,

auch die Differenz $c_1 - c_2 \in C$. Es ist leicht einzusehen, dass

$$dist(c_1, c_2) = dist(c_1 - c_2, \vec{0}) = gew(c_1 - c_2)$$

gilt. Der Hammingabstand zweier verschiedener Codeworte aus C läßt sich also stets auch durch das Gewicht eines dritten, von Null verschiedenen, Codevektors aus C ausdrücken; umgekehrt gilt dies natürlich insbesondere, das Gewicht eines Codevektors ist ja gerade dessen Hammingabstand vom Nullvektor. Also gilt folgender

Satz 1.2 *Für jeden linearen Code* $C \leq \mathbb{F}_q^n$ *ist* $dist(C) = min\{gew(c) : c \in C^*\}$.

□

Statt die Hammingdistanz aller Paare aus C zu bestimmen (Aufwand quadratisch in $|C|$), kommt man demnach für die Bestimmung von $dist(C)$ bei linearen Codes mit linearem Aufwand in $|C|$ zurande; es existieren sogar Algorithmen, bei denen im Normalfall längst nicht alle $c \in C$ betrachtet werden müssen, vgl. z.B. [BBF$^+$06, S. 70ff.].

Ein weiterer Vorteil linearer Codes ist die Möglichkeit, sie auf sehr kurze Weise beschreiben zu können: Es genügt, eine Basis anzugeben. Die Basisvektoren eines linearen Codes C schreibt man dazu in eine Matrix Γ; verwendet man dabei, wie in der Codierungstheorie üblich, die Zeilenkonvention, so ist $\Gamma \in \mathbb{F}_q^{k \times n}$ (für $dim(C) = k$ und Blocklänge n). Eine solche Matrix nennt man eine *Generatormatrix* von C. Dann ist $C = \{v\Gamma : v \in \mathbb{F}_q^k\}$ und für gewöhnlich wählt man die Codierungsfunktion γ auch als die durch Γ bestimmte, lineare Abbildung, also:

$$\gamma : \mathbb{F}_q^k \rightarrowtail \mathbb{F}_q^n, v \mapsto v\Gamma\,.$$

Natürlich gibt es zu demselben linearen Code im Allgemeinen viele verschiedene Generatormatrizen - die Darstellung ist also längst nicht eindeutig. Der in 1.4 angegebene Beispielcode ist z. B. ein dreidimensionaler linearer Code über $\mathbb{F}_2$ der Blocklänge sechs und *eine* Generatormatrix dazu ist

$$\Gamma = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Durch $\gamma(v) = v\Gamma$ erhält man dann auch die in 1.4 verwendete Codierung.

Noch ein Punkt, der für lineare Codes spricht, ist die Möglichkeit einer schnellen Decodierung nach dem MLD-Prinzip; hier sei nur das Stichwort *Syndromdecodierung* genannt. Damit läßt sich der Aufwand, einen empfangenen Vektor $\tilde{c} \in \mathbb{F}_q^n$ in das hinsichtlich Hammingdistanz nächstgelegene Codewort $\bar{c} \in C$ zu decodieren, im Wesentlichen auf eine Matrixmultiplikation $\tilde{c}\Delta^t$ beschränken, welche das *Syndrom* von $\tilde{c}$ liefert; Δ ist dabei eine sogenannte *Kontrollmatrix* von C. Näheres dazu findet sich in [BBF$^+$06, S. 24f.].

1.6 Äquivalenz von Codes

Wie schon erwähnt, gibt es zu ein und demselben linearen Code gewöhnlich sehr viele verschiedene Generatormatrizen; genauer kann man es so formulieren:

Satz 1.3 *Zwei Generatormatrizen $\Gamma_1, \Gamma_2 \in \mathbb{F}_q^{k\times n}$ vom Rang k erzeugen genau dann denselben linearen Code C, wenn es eine invertierbare Matrix $B \in \mathbb{F}_q^{k\times k}$ gibt, s. d. $\Gamma_2 = B\Gamma_1$.*

Beweis:
„$\Leftarrow$“: Sei $\Gamma_2 = B\Gamma_1$ mit invertierbarer Matrix $B \in \mathbb{F}_q^{k\times k}$. Dann ist

$$\{v\Gamma_1 : v \in \mathbb{F}_q^k\} = \{vB^{-1}B\Gamma_1 : v \in \mathbb{F}_q^k\} = \{\underbrace{(vB^{-1})}_{\tilde{v}}\Gamma_2 : v \in \mathbb{F}_q^k\} = \{\tilde{v}\Gamma_2 : \tilde{v} \in \mathbb{F}_q^k\}.$$

Also erzeugen Γ_1 und Γ_2 denselben linearen Code.

„$\Rightarrow$“: Wenn Γ_1 und Γ_2 denselben linearen Code C erzeugen, so bilden die Zeilen von Γ_1 bzw. Γ_2 jeweils eine Basis von C. Insbesondere läßt sich jede Zeile $\gamma_i^{(2)}$ von Γ_2 linear aus den Zeilen von Γ_1 kombinieren, etwa $\gamma_i^{(2)} = b_i\Gamma_1$ mit $b_i \in \mathbb{F}_q^k\,(i = 1, 2, ..., k)$. Dann gilt für die Matrix

$$B := \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{k-1} \\ b_k \end{pmatrix} :$$

$\Gamma_2 = B\Gamma_1$. Da B außerdem eine Basis von C auf eine andere Basis abbildet, ist B invertierbar.

□

Aber auch formal verschiedene lineare Codes zu demselben Parametersatz (n, k, d, q) können unter gewissen Voraussetzungen als äquivalent angesehen werden:

Für $\pi \in S_n$ und $u = (u_1, u_2, ..., u_n) \in \mathbb{F}_q^n$ sei

$$\pi(u) := \left(u_{\pi^{-1}(1)}, u_{\pi^{-1}(2)}, ..., u_{\pi^{-1}(n)}\right).$$

Zu einem linearen Code C und $\pi \in S_n$ werde nun mit C_π der Code bezeichnet, der durch Anwenden von π auf die Codeworte von C entsteht, d.h. $C_\pi = \{\pi(c) : c \in C\}$. C und C_π sind zwar im Allgemeinen nicht identisch, hinsichtlich ihrer Korrektureigenschaften aber

offenbar vollkommen gleichwertig: Schließlich ist die Hammingdistanz zweier Codeworte in C immer gleich der Hammingdistanz ihrer Bilder unter π:

$$\forall c_1, c_2 \in C : dist(c_1, c_2) = dist(\pi(c_1), \pi(c_2)) .$$

Multipliziert man zusätzlich noch jeweils die i-te Koordinate der Worte aus $\mathbb{F}_q^n$ mit einem Element $\lambda_i \in \mathbb{F}_q^*$, was, mit $\lambda := (\lambda_1, \lambda_2, ..., \lambda_n)$, durch die Abbildung

$$\mu_\lambda : \mathbb{F}_q^n \to \mathbb{F}_q^n, \quad (u_1, u_2, .., u_n) \mapsto (\lambda_1 u_1, \lambda_2 u_2, ..., \lambda_n u_n)$$

beschrieben werden kann, so ist der entstehende Code

$$C_{\mu_\lambda \circ \pi} := \{\mu_\lambda(\pi(c)) : c \in C\}$$

in obigem Sinne immer noch äquivalent zu C, denn wiederum gilt:

$$\forall c_1, c_2 \in C : dist(c_1, c_2) = dist(\mu_\lambda(\pi(c_1)), \mu_\lambda(\pi(c_2))).$$

Jede solche Abbildung $\mu_\lambda \circ \pi$ entspricht dabei eineindeutig einer Rechtsmultiplikation der Elemente aus $\mathbb{F}_q^n$ mit einer Matrix $M(\mu_\lambda, \pi)$ aus der *vollen monomialen Gruppe* $M_n(q)$, welche gerade aus den Matrizen in $\mathbb{F}_q^{n \times n}$ besteht, bei denen in jeder Zeile und jeder Spalte jeweils *genau* ein Nichtnullelement aus $\mathbb{F}_q$ auftritt. Konkret hat $M(\mu_\lambda, \pi)$ folgende Gestalt:

$$M(\mu_\lambda, \pi) = \begin{pmatrix} & & & k & & & \\ & & & \downarrow & & & \\ & & & 0 & & & \\ & & & \vdots & & & \\ & & & 0 & & & \\ 0 & \dots & 0 & \lambda_k & 0 & \dots & 0 \\ & & & 0 & & & \\ & & & \vdots & & & \\ & & & 0 & & & \end{pmatrix} \leftarrow i = \pi^{-1}(k)$$

Eine Abbildung $\iota : \mathbb{F}_q^n \to \mathbb{F}_q^n$ mit der Eigenschaft:

$$\forall u, v \in \mathbb{F}_q^n : dist(u, v) = dist(\iota(u), \iota(v))$$

nennt man *(globale) Isometrie*, ist ι zusätzlich linear, spricht man entsprechend von einer *(globalen) linearen Isometrie*. Es läßt sich zeigen, dass jede globale lineare Isometrie die zugehörige lineare Abbildung zu genau einer Matrix aus $M_n(q)$ ist [BBF+06, S. 30ff.]. Umgekehrt beschreibt natürlich, wie gesehen, jede Matrix aus $M_n(q)$ eine globale lineare Isometrie. Weiterhin gilt sogar, dass sich eine beliebige *lokale lineare Isometrie* zwischen zwei linearen (n, k, q)-Codes C_1 und C_2, also eine lineare Abbildung

$$\bar{\iota} : C_1 \to C_2, \; s.d. \, \forall c_1, c_2 \in C_1 : dist(c_1, c_2) = dist(\bar{\iota}(c_1), \bar{\iota}(c_2)),$$

immer zu einer globalen Isometrie fortsetzen läßt [BBF+06, S. 549ff.]. Deswegen ist in der folgenden Definition auch egal, ob eine lokale oder globale Isometrie gefordert wird:

Definition 1.1 *Zwei lineare Codes C_1 und C_2 heißen* linear isometrisch*, wenn es eine (lokale oder globale) lineare Isometrie ι gibt, so dass $\iota(C_1) = C_2$.*

□

Ist C ein linearer Code, so nennen wir die Menge aller zu C linear isometrischen Codes die *lineare Isometrieklasse von C* (aus gruppentheoretischer Sicht kann diese auch als die Bahn von C unter der vollen monomialen Gruppe interpretiert werden); die Menge aller linearen Isometrieklassen zu k-dimensionalen Codes der Blocklänge n über $\mathbb{F}_q$ ist dann eine Partition von $U(n,k,q) := \{U \leq \mathbb{F}_q^n : dim(U) = k\}$.

Eine Verallgemeinerung der linearen Isometrien sind die *semilinearen Isometrien*. Eine globale semilineare Isometrie σ hat die Gestalt $\sigma = (\alpha, \iota)$. Dabei ist α ein Automorphismus von $\mathbb{F}_q$ und ι eine globale lineare Isometrie. σ permutiert die Elemente aus $\mathbb{F}_q^n$ wie folgt:

$$\sigma : \mathbb{F}_q^n \to \mathbb{F}_q^n,\ \sigma((u_1, u_2, ..., u_n)) = \iota((\alpha(u_1), \alpha(u_2), ..., \alpha(u_n))).$$

Genau wie bei den linearen Isometrien kann man auch jede lokale semilineare Isometrie zwischen zwei linearen Codes zu einer globalen fortsetzen [BBF$^+$06, S. 552f.]; außerdem läßt sich, für $n \geq 3$, jede beliebige globale Isometrie auf $\mathbb{F}_q^n$, die Unterräume auf Unterräume abbildet, als semilineare Isometrie schreiben [BBF$^+$06, S. 45ff.]. Und ebenso wie bei den linearen Isometrien definieren wir:

Definition 1.2 *Zwei lineare Codes C_1 und C_2 heißen* semilinear isometrisch*, wenn es eine (lokale oder globale) semilineare Isometrie σ gibt, so dass $\sigma(C_1) = C_2$.*

□

Auch die Definition der *semilinearen Isometrieklasse* eines linearen Codes ist vollkommen analog.

Ob nun linear bzw. semilinear isometrische Codes auch tatsächlich „äquivalent“ im Sinne des Wortes sind, hängt vom Kontext ab, in dem man sie untersucht; wir werden in Abschnitt 4.4 darauf zu sprechen kommen, wie es diesbezüglich für die in dieser Arbeit vorgestellten Algorithmen aussieht.

Kapitel 2

Vorüberlegungen für einen heuristischen Konstruktionsalgorithmus

In diesem Abschnitt wird, in relativ groben Zügen, der in ***„Heurico“*** verwendete Ansatz für einen heuristischen Konstruktionsalgorithmus von linearen Codes vorgestellt. Details und Verbesserungen werden dann in den beiden folgenden Kapiteln näher ausgeführt.

2.1 Beobachtung zur Gewichtssumme linearer Codes

Ein k-dimensionaler linearer Code C über $\mathbb{F}_q$ besteht, unabhängig von n, stets aus genau q^k verschiedenen Codeworten. Diese ergeben sich durch Multiplikation der Elemente aus $\mathbb{F}_q^k$ mit einer Generatormatrix Γ: $C = \{v\Gamma : v \in \mathbb{F}_q^k\}$. Dabei wird die j-te Koordinate der Codevektoren allein durch die j-te Spalte γ^j von Γ bestimmt:

$$c = (c_1, c_2, ..., c_n) = v\Gamma \Rightarrow \forall j = 1, ..., n : c_j = v\gamma^j.$$

Weiterhin gilt:

Lemma 2.1 *Ist Γ die Generatormatrix eines k-dimensionalen, linearen Codes C und die j-te Spalte γ^j von Γ eine Nichtnullspalte, so gibt es genau $q^{k-1}(q-1)$ Vektoren in C, deren j-te Koordinate ungleich Null ist.*

Beweis:
Für $c = (c_1, c_2, ..., c_n) \in C$ mit $c = v\Gamma$ und $\overline{\gamma}^j := (\gamma^j)^t$ ist

$$c_j = 0 \Leftrightarrow v\gamma^j = 0 \Leftrightarrow v \in \langle\overline{\gamma}^j\rangle^\perp.$$

Damit folgt:

$$|\{c = (c_1, c_2, ..., c_n) \in C : c_j = 0\}| = |\langle \overline{\gamma}^j \rangle^\perp| = q^{k-1},$$

denn $\langle \overline{\gamma}^j \rangle^\perp$ ist ein $(k-1)$-dimensionaler Unterraum von $\mathbb{F}_q^k$. Deshalb:

$$|\{c = (c_1, c_2, ..., c_n) \in C : c_j \neq 0\}| = q^k - q^{k-1} = q^{k-1}(q-1).$$

□

Da, für eine Nullspalte von Γ, die entsprechende Koordinate *aller* Codeworte in C Null ist, folgt unmittelbar

Lemma 2.2 *Ist $\Gamma \in \mathbb{F}_q^{k \times n}$ die Generatormatrix eines k-dimensionalen, linearen Codes C der Blocklänge n und enthält Γ genau m Nullspalten, so gilt für die Summe der Gewichte aller Codeworte aus C:*

$$\sum_{c \in C} gew(c) = (n-m)q^{k-1}(q-1)\,.$$

□

Lineare Codes, deren Generatormatrizen keine Nullspalte enthalten, wollen wir *nichtredundant* nennen. Offensichtlich können wir uns bei der Suche nach Codes mit möglichst hoher Minimaldistanz auf die nichtredundanten beschränken. Lemma 2.2 liefert für diesen Fall ($m = 0$):

Lemma 2.3 *Für einen nichtredundanten, linearen Code C mit Blocklänge n ist*

$$\sum_{c \in C} gew(c) = nq^{k-1}(q-1)\,.$$

□

2.2 Das heuristische Grundprinzip: lokal beschränktes Vorausdenken

Ab jetzt wollen wir für den Rest der Arbeit davon ausgehen, dass uns die Ordnung q des Grundkörpers sowie die Dimension k, die Blocklänge n und eine untere Schranke d für

die Minimaldistanz des gewünschten linearen Codes fest vorgegeben sind; die Aufgabe lautet also:

„Konstruiere die Generatormatrix Γ eines irredundanten, linearen (n, k, d', q)-Codes C, wobei $d' \geq d$“.

Aus Lemma 2.3 wissen wir, dass, egal wie wir Γ wählen, die Summe der Gewichte aller Codeworte stets $nq^{k-1}(q-1)$ betragen wird. Die Minimaldistanz von C ist aber, nach Satz 1.2, gerade das kleinste unter allen Codeworten in C^* auftretende Gewicht. Es gilt also, dieses minimale Gewicht möglichst groß zu halten. Weil jedes Codewort mit „hohem“ Gewicht (aufgrund der festen Gewichtssumme) den anderen Codeworten quasi Gewicht „wegnimmt“ (und deshalb unter diesen tendenziell auch eines mit „niedrigem“ Gewicht auftreten wird), wäre es ideal, wenn die „Gewichtsmasse“ $nq^{k-1}(q-1)$ gleichmäßig auf die $q^k - 1$ Codeworte in C^* verteilt würde. Nun ist aber das arithmetische Mittel

$$\rho := \frac{nq^{k-1}(q-1)}{q^k - 1}$$

im Allgemeinen keine natürliche Zahl, in diesem Fall ist eine absolut homogene Gewichtsverteilung unmöglich. Aber es muss für mindestens ein Codewort $c \in C^*$ gelten: $gew(c) \leq \rho$. Denn andernfalls wäre

$$\sum_{c \in C} gew(c) \overset{gew(\vec{0})=0}{=} \sum_{c \in C^*} gew(c) > \sum_{c \in C^*} \rho = \sum_{c \in C^*} \frac{nq^{k-1}(q-1)}{q^k - 1} \overset{|C^*|=q^k-1}{=} nq^{k-1}(q-1),$$

ein direkter Widerspruch zu Lemma 2.3. Damit folgt unter anderem die sogenannte Plotkinschranke, denn die Existenz eines Codeworts c mit $gew(c) \leq \rho$ gilt natürlich insbesondere auch für redundante lineare Codes:

Satz 2.1 (Plotkinschranke) *Für jeden linearen (n, k, d, q)-Code gilt:*

$$d \leq \left\lfloor \frac{nq^{k-1}(q-1)}{q^k - 1} \right\rfloor .$$

□

Aber wie kann man nun versuchen, eine Generatormatrix für einen Code mit möglichst hoher Minimaldistanz zu finden? Der naive brute-force-Ansatz, einfach alle möglichen Spaltenauswahlen durchzuprobieren, wird schnell durch eine Überschlagsrechnung ad absurdum geführt: Zwar braucht man aus jeder Äquivalenzklasse von $\mathbb{F}_q^{k^*}$ bzgl. linearer Abhängigkeit nur einen Repräsentanten zu berücksichtigen, kann sich also auf die *projektive Geometrie*

$$PG_{k-1}(q) := \{\langle v \rangle : v \in \mathbb{F}_q^{k^*}\}$$

beschränken (denn jede Spalte in einer Generatormatrix läßt sich durch ein Vielfaches ihrer selbst ersetzen, ohne dass sich die Gewichte und Abstände der erzeugten Codeworte ändern); aber selbst, wenn man nur Generatormatrizen projektiver Codes untersucht (d.h. je zwei Spalten sind linear unabhängig), gibt es

$$\binom{\frac{q^k-1}{q-1}}{n}$$

Möglichkeiten, eine im Regelfall astronomisch große Zahl. Also muss man sich irgendwie anders behelfen. Als Motivation für die hier gewählte Methode soll ein zugegebenermaßen etwas gewagter Vergleich dienen:

Auch das Schachspiel ist eine sehr komplizierte Angelegenheit - in einer gewöhnlichen Mittelspielstellung hat jede Seite normalerweise mindestens 40 Zugmöglichkeiten. Ginge man nun davon aus, dass eine Partie durchschnittlich nach je 60 Zügen beider Parteien beendet wäre (was bei perfektem Spiel vermutlich mindestens um den Faktor 5 untertrieben sein dürfte), so ergäben sich ohne Zugumstellungen etwa 40^{120} verschiedene Partieabläufe, die berechnet werden müssten. Zwar erlaubt der sogenannte „Alpha-Beta-Algorithmus“, diese Zahl im Idealfall ungefähr auf ihre Wurzel zu reduzieren, dennoch liegt eine Aufgabe solchen Umfangs weit jenseits des für Computer Berechenbaren. Und trotzdem spielen heutige Spitzenprogramme bereits auf handelsüblichen Rechnern stärker als die besten menschlichen Vertreter. Ihre Vorgehensweise bei der Zugfindung dürfte im Wesentlichen stets dieselbe sein: Da ein „Ausrechnen“ der Stellung praktisch unmöglich ist, werden einfach alle Möglichkeiten bis zu einer gewissen Tiefe ($\widehat{=}$ Zugzahl ab der Ausgangsstellung) durchprobiert und für die jeweils entstandene Stellung mit einer Bewertungsfunktion *abgeschätzt*, welche Partei besser steht und in welchem Maße. Anhand dieser Berechnungen kann das Programm dann eine Entscheidung treffen, welcher Zug ihm in der Ausgangsposition die (gemäß dieser Heuristik) besten *Chancen* bietet, die Partie für sich erfolgreich zu beenden. Eine für eine vollständige Berechnung zu komplexe Aufgabe kann also unter Umständen immer noch zufriedenstellend gelöst werden, indem man sie in mehrere einzelne, aufeinanderfolgende Entscheidungsprobleme zerlegt ($\widehat{=}$ Wahl des jeweils *nächsten* Zuges im Schach) und bei diesen Unterproblemen den Umfang der Berechnungen derart beschränkt, dass sie in praktisch vernünftiger Zeit lösbar sind ($\widehat{=}$ Begrenzung der Suchtiefe und Abschätzung durch Bewertungsfunktion).

Natürlich ist Schachspielen etwas anderes als die Konstruktion linearer Codes - beispielsweise gibt es im Gegensatz zum Schach in der Codierungstheorie keinen Gegenspieler. Trotzdem kann man versuchen, einen Teil des Ansatzes zu retten: In unserem Falle könnte man die Wahl der nächsten Spalte für die Generatormatrix mit der Wahl des nächsten Zuges im Schach vergleichen. Wir setzen

$$M := M_{(n,k,q)} := \{\Gamma \in \mathbb{F}_q^{k\times m} : m \leq n\}$$

und wollen zunächst davon ausgehen, dass uns schon ein boolescher Vergleichsoperator

$$>_{(n,k,d,q)}\colon M \times M \to \{true, false\}$$

gegeben ist, dessen Wert $>_{(n,k,d,q)}(\Gamma_1, \Gamma_2)$ angeben soll, ob sich Γ_1 durch Hinzunahme von Spalten (unter heuristischen Gesichtspunkten) mit „höherer Wahrscheinlichkeit“ zur Generatormatrix eines $(n, k, \geq d, q)$-Codes erweitern lässt als Γ_2.

Anstelle von $>_{(n,k,d,q)}(\Gamma_1, \Gamma_2) = \mathit{true}$ schreiben wir auch einfach $\Gamma_1 >_{(n,k,d,q)} \Gamma_2$ und lassen dabei, wenn sie aus dem Zusammenhang klar sind, auch noch die Parameter weg, notieren also nur noch $\Gamma_1 > \Gamma_2$. Mit $(\Gamma \,|\, \gamma)$ bezeichnen wir die Matrix, die sich aus Γ durch rechtsseitiges Anfügen der Spalte γ ergibt (passende Dimensionen von Γ und γ vorausgesetzt). Dann wäre ein simpler Algorithmus:

Eingabe : **int** n, k, d, q: gewünschte Parameter des linearen Codes
MATRIX $\Gamma_0 \in \mathbb{F}_q^{k \times n_0}$: vorgegebene Ausgangsmatrix $(n_0 \leq n)$
Ausgabe : Generatormatrix Γ zu $(n, k, \geq d, q)$-Code oder FAILED

```
1  PROCEDURE main() : MATRIX bzw. FAILED
2  {
3    MATRIX Γ ← Γ_0;
4    for int m from n_0 to n − 1 do
5      COLUMN γ ←⊥;
6      foreach ⟨v⟩ ∈ PG_{k−1}(q) do
7        if γ = ⊥ or (Γ | v^t) > (Γ | γ) then
8          γ ← v^t;
9        endif
10     endfch
11     Γ ← (Γ | γ);
12   endfor
13   if dist(LinearCode(Γ)) ≥ d then
14     return Γ;
15   endif
16   return FAILED;
17 }
```

Algorithmus 1 : Basisalgorithmus

In Zeile 3 wird die Matrix Γ, die am Ende ggf. den Code mit den gewünschten Parametern generieren soll, zunächst mit der übergebenen Startmatrix gleichgesetzt. Sinnvolle Vorgaben sind z. B. die k-dimensionale Einheitsmatrix I_k (jeder lineare Code C ist linear isometrisch zu einem systematisch codierten Code C', vgl. [BBF$^+$06, S. 65f.]) oder die Generatormatrix eines (n_0, k, d_0, q)-Codes mit relativ hoher Minimaldistanz d_0. Es kann sich aber auch einfach um die „leere“ Matrix handeln, in diesem Fall ist der Algorithmus eben von Anfang an auf sich selbst gestellt. In der darauffolgenden Schleife (Z. $4 - 12$) werden nun sukzessive Spalten hinzugenommen, wobei der Schleifenzähler m anzeigt, wieviel Spalten bei Eintritt in die Schleife bereits gewählt waren. Zu Beginn (Z. 5) wird zunächst die Variable γ, die am Schleifenende die nächste hinzuzufügen-

de Spalte enthält, mit einem Wert initialisiert, der *keiner* zulässigen Spalte entspricht (Symbol $\perp$). In den Zeilen $6-10$ steht eine zweite Schleife, die über alle möglichen Repräsentanten $\langle v\rangle \in PG_{k-1}(q)$ läuft und dabei jeweils prüft, ob die entstehende Matrix $(\Gamma \mid v^t)$ günstiger erscheint als der bisher beste „Kandidat“ $(\Gamma \mid \gamma)$. Ist dies der Fall (bzw. $\gamma = \perp$, d. h. die Schleife wird erstmalig durchlaufen), so wird v^t zur neuen besten Spalte. Gab es zwei oder mehr gleichwertige, „beste“ Spalten, so enthält γ diejenige, die zuerst untersucht wurde. Das genaue Ergebnis hängt unter Umständen also auch von der in Zeile 6 gewählten, hier nicht näher spezifizierten Durchlaufreihenfolge ab. In Zeile 11 wird dann γ an Γ angefügt und die Suche beginnt erneut für die nächste Spalte. Besteht Γ schließlich aus n Spalten, so wird die äußere Schleife verlassen und in den Zeilen $13-16$ das Ergebnis ausgewertet.

Im nächsten Kapitel wollen wir uns der Implementierung des Operators „$>$“ widmen.

Kapitel 3

Der Vergleichsoperator „$>$“

Wie schon gegen Ende des vorherigen Kapitels soll, zu gegebenen Parametern n, k, d, q, die Menge M durch

$$M := M_{(n,k,q)} := \{\Gamma \in \mathbb{F}_q^{k\times m} : m \leq n\}$$

definiert sein. Wir beschäftigen uns in diesem Kapitel mit Möglichkeiten zur Implementierung eines Vergleichsoperators „$>$“ (genauer $>_{(n,k,d,q)}$) mit

$$>: M \times M \to \{true, false\},$$

dessen boolescher Wert $>(\Gamma_1, \Gamma_2)$ angibt, ob die „Aussichten“, Γ_1 zur Generatormatrix eines (n, k, d, q)-Codes erweitern zu können, besser sind als die für Γ_2. Wenn dies zutrifft, schreiben wir kurz: $\Gamma_1 > \Gamma_2$. Es sollen hierfür zwei Alternativen vorgestellt werden, die beide im Programm ***„Heurico“*** des Verfassers mit gutem Erfolg getestet wurden. Auf eine dritte, naheliegende Möglichkeit, die sich jedoch aus einem interessanten Grund als für unsere Zwecke unbrauchbar erweist, wird in einem Exkurs in Anhang A eingegangen.

3.1 „Lexikographische“ Minimierung der Gewichtsverteilung

Die Minimaldistanz eines linearen Codes C wird durch die gewichtsminimalen Codeworte in C festgelegt. Da durch Hinzufügen einer Spalte γ zu einer Generatormatrix Γ die von Γ erzeugten Codeworte nur um eine Komponente erweitert und ansonsten nicht verändert werden, bleibt die Minimaldistanz des erzeugten Codes hierbei entweder gleich oder erhöht sich um eins. Hat man also zwei verschiedene Generatormatrizen Γ_1, Γ_2 gleicher Dimension, etwa $\Gamma_1, \Gamma_2 \in \mathbb{F}_q^{k\times m}$, zu den linearen Codes C_1 bzw. C_2 gegeben, und ist $dist(C_1) = dist(C_2) + 1$, so gilt für die nach Hinzufügen von je einer Spalte γ_1 bzw. γ_2

entstehenden Matrizen $\hat{\Gamma_1} := (\Gamma_1|\gamma_1)$ und $\hat{\Gamma_2} := (\Gamma_2|\gamma_2)$ immer noch: $dist(\hat{\Gamma_1}) \geq dist(\hat{\Gamma_2})$, wobei $dist(\Gamma)$ für eine Generatormatrix Γ hier wie im Folgenden stets die Minimaldistanz des von Γ erzeugten, linearen Codes meint. Ist $dist(C_1) = dist(C_2) + i$ für ein $i \in \mathbb{N}$, so gilt dies sogar nach Hinzunahme von je i beliebigen Spalten zu Γ_1 und Γ_2. Aber es könnte natürlich auch sein, dass sich der „Vorsprung“ von $\hat{\Gamma_1}$ im Vergleich zu $\hat{\Gamma_2}$ gegenüber den Ausgangsmatrizen bei jeweiliger Hinzunahme der beiderseits günstigsten Spalten sogar noch vergrößert. *Tendenziell* (keinesfalls *immer*) entstehen also aus Generatormatrizen zu Codes hoher Minimaldistanz durch Hinzunahme weiterer Spalten auch wieder eher Generatormatrizen „guter“ Codes. Damit hätten wir schon eine Teildefinition von $>$:

$$dist(\Gamma_1) > dist(\Gamma_2) \Rightarrow \Gamma_1 > \Gamma_2$$

bzw.

$$dist(\Gamma_1) < dist(\Gamma_2) \Rightarrow \Gamma_1 \leq \Gamma_2$$

($\Gamma_1 \leq \Gamma_2$ steht im zweiten Fall verkürzend für die Aussage $> (\Gamma_1, \Gamma_2) = false$).

Man mag nun einwenden, dass in dieser Definition keinerlei Rücksicht auf die Dimension der Generatormatrizen genommen wird (beispielsweise ist es sehr zweifelhaft, ob man im Allgemeinen einen linearen Code mit Minimaldistanz 5 bei Blocklänge 50 wirklich als „besser“ ansehen möchte als einen mit Minimaldistanz 4 bei Blocklänge 10). Da aber in Algorithmus 1 (und allen in dieser Arbeit vorgestellten Varianten) ohnehin nur Matrizen gleicher Dimension verglichen werden, spielt dieser Aspekt keine Rolle - was keinesfalls bedeutet, dass nicht Verfahren denkbar sind, in denen ein (sinnvoller) Vergleich von Matrizen unterschiedlicher Dimension wünschenswert wäre.

Was jetzt noch aussteht, ist die Definition von $>$ für den Fall $dist(\Gamma_1) = dist(\Gamma_2)$. Hier bietet es sich an, jeweils die Codeworte vom Minimalgewicht zu zählen und die Matrix mit der geringeren Anzahl als „besser“ anzusehen. Sollten gleichviele Codeworte zum Minimalgewicht existieren, so führt man denselben Vergleich für Minimalgewicht+1 durch, herrscht auch hier Gleichheit, dann für Minimalgewicht+2, usw.. Nur wenn die Anzahlen für alle Gewichte gleich sind, werden auch die Matrizen als gleichwertig angesehen. Unter Verwendung von

$$A_i^j := |\{v \in \mathbb{F}_q^k : gew(v\Gamma_j) = i\}| \qquad (j \in \{1, 2\})$$

lautet die formale Definition für $>$:

$$\Gamma_1 > \Gamma_2 :\Leftrightarrow \exists i_0 \in \mathbb{N}, i_0 \leq n : (\forall i < i_0 : A_i^1 = A_i^2 \wedge A_{i_0}^1 < A_{i_0}^2)\,.$$

3.2 Approximation der Erfolgschance bei zufälliger Spaltenhinzunahme

Ein Problem bei Verwendung des in 3.1 vorgestellten Vergleichsoperators ist, dass dabei „mit Gewalt“ in jedem Schritt die Minimaldistanz maximiert (und die Anzahl der ent-

sprechenden Codeworte minimiert) wird - (fast) ohne Rücksicht auf die Codeworte von nichtminimalem Gewicht. Das kann alles andere als optimal sein. Schließlich muss, um am Ende Minimaldistanz $\geq d$ zu erreichen, jedes Codewort c, für dessen Gewicht $gew(c)$ nach dem i-ten Schritt

$$gew(c) < d$$

gilt, durch die Hinzunahme weiterer $n - i$ Spalten zur Generatormatrix um mindestens $d - gew(c)$ Nichtnullkomponenten erweitert werden, ist also ein potentieller „Stolperstein“. Unter Umständen ist es besser, die Minimaldistanz in einem Schritt nicht zu erhöhen, wenn dafür viele Codeworte von nur leicht höherem Gewicht eine weitere Nichtnullkomponente erhalten. Dies soll an einem konkreten Beispiel erläutert werden:

Wir betrachten den Fall $n = 8$, $k = 4$, $q = 2$. Die optimale Minimaldistanz für einen linearen Code ist hier $d = 4$ und nach [Bet] gibt es hierfür nur eine einzige lineare Isometrieklasse. Die Codes dieser Klasse bestehen neben dem Nullvektor aus 14 Codeworten vom Gewicht vier und einem Codewort mit Gewicht acht. Eine systematische Generatormatrix ist

$$\Gamma_0 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

Wir wollen nun davon ausgehen, dass wir Algorithmus 1 mit dem Vergleichsoperator nach 3.1 und den oben genannten Parametern anwenden und anstelle der ersten vier Schritte die Einheitsmatrix I_4 vorgeben - da jeder lineare Code linear isometrisch zu einem Code mit systematischer Generatormatrix ist, kann man sich hierdurch noch nichts „verbauen“. Der so entstehende $(4, 4, 1, 2)$-Code besteht gerade aus allen Elementen von $\mathbb{F}_2^4$, hat also die Gewichtsverteilung $[1 - 4]\,(4, 6, 4, 1)$ (lies: „Minimales Gewicht ist eins, maximales vier, es gibt vier Codeworte mit Gewicht eins, sechs mit Gewicht zwei, ...“). Der Algorithmus würde jetzt einen Paritätscheck durchführen, das entspricht bei Vorliegen einer Einheitsmatrix dem Anfügen der Spalte aus lauter Einsen - denn damit wird nach dem fünften Schritt Minimaldistanz zwei erreicht, bei Hinzunahme irgendeiner anderen Spalte verbliebe in der Generatormatrix stets mindestens eine Zeile (und deshalb auch ein Wort im erzeugten Code) vom Gewicht eins. Die genaue Gewichtsverteilung nach Hinzunahme dieser Spalte ist $[2 - 5]\,(10, 0, 5)$. Es wurde also, im Vergleich zum vorhergehenden Schritt, das Gewicht der vier Codeworte von Gewicht eins inkrementiert, im Gegenzug aber wurden die sechs Codeworte von Gewicht zwei um eine Null erweitert, so dass jetzt noch zehn „Problemkandidaten“ verbleiben, deren Gewicht mit den nächsten drei Spalten um jeweils mindestens zwei gesteigert werden müsste. Aber dies ist nicht möglich, wie eine einfache Überlegung zeigt:

Das Endergebnis nach Festlegung der achten Spalte wäre auf jeden Fall eine Matrix Γ

der Gestalt $\Gamma = (\Gamma_1|\Gamma_2)$ mit

$$\Gamma_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \textit{ und } \Gamma_2 \in \mathbb{F}_2^{4\times 3}\,.$$

Hätte der von Γ erzeugt Code Minimaldistanz vier, so wäre, aufgrund der Eindeutigkeit der oben erwähnten Isometrieklasse, eines seiner Codeworte dasjenige vom maximalen Gewicht acht, also $c_0 = (11111111)$. Es müsste demnach ein $v \in \mathbb{F}_2^4$ geben mit $v\Gamma = c_0$. Das impliziert aber insbesondere $vI_4 = (1111)$ und damit $v = (1111)$. Für dieses v ist aber die fünfte Komponente von $v\Gamma$ eine Null. Γ kann also (unabhängig vom genauen Aussehen von Γ_2!) niemals c_0 und somit auch keinen Code aus *der* Isometrieklasse zur Minimaldistanz vier erzeugen. Somit haben wir ein Beispiel gesehen, in dem das „gierige“ schrittweise Maximieren der Minimaldistanz die Konstruktion eines optimalen Codes bereits nach dem ersten (eigenständigen) Schritt unmöglich macht. Genau dieser Umstand motiviert die folgenden Überlegungen.

Beim Vergleich zweier Generatormatrizen Γ_1 und Γ_2 sollten alle Codeworte Berücksichtigung finden, deren Gewicht noch unterhalb der angestrebten Minimaldistanz d liegt - wobei natürlich naheliegt, den Codeworten mit niedrigerem Gewicht eine größere Bedeutung beizumessen. Ein Ansatz, der diesem Rechnung trägt, wird jetzt vorgestellt:

Wir haben in 2.1 gesehen, dass es für jede Koordinate (zu einer Nichtnullspalte) genau $q^{k-1}(q-1)$ Codeworte gibt, die an dieser Stelle von Null verschieden sind. Wählt man also ein beliebiges $v \in \mathbb{F}_q^{k^*}$ und eine zufällige Spalte γ (γ^t ebenfalls aus $\mathbb{F}_q^{k^*}$), so ist die Wahrscheinlichkeit, dass deren Skalarprodukt ungleich Null ist, gerade:

$$p := P(\langle v, \gamma^t\rangle \neq 0) = \frac{(q-1)q^{k-1}}{q^k - 1}\,.$$

Entsprechend ist die Wahrscheinlichkeit $r(i,j)$, dass für ein solches v von i zufällig gewählten Nichtnullspalten $\gamma_1, \gamma_2, ..., \gamma_i$ genau j mit v ein Skalarprodukt ungleich Null haben, gerade

$$r(i,j) = p^j(1-p)^{i-j}\binom{i}{j}\,.$$

Mit $s(i,j)$ wollen wir die Wahrscheinlichkeit bezeichnen, dass in der gerade skizzierten Situation *mindestens* j Spalten bei Skalarmultiplikation mit v einen von Null verschiedenen Wert liefern. Sie ist dann

$$s(i,j) = \sum_{l=j}^{i} r(i,l)\,.$$

Wofür ist das jetzt gut? Nun, stellen wir uns unter $\gamma_1, \gamma_2, ..., \gamma_{n-m}$, mit $\gamma_i^t \in \mathbb{F}_q^{k^*}$, zufällig gewählte Nichtnullspalten vor, die rechtsseitig an Γ angefügt werden; den Code, der

von der dabei entstehenden Matrix erzeugt wird, wollen wir mit $C_\Gamma^{(n)}$ bezeichnen. Alle nun folgenden stochastischen Erwägungen beziehen sich auf die Phase *vor* der zufälligen Festlegung der Spalten.

Mit den vorangegangenen Überlegungen können wir auf sehr einfache Weise eine *Approximation* für die Wahrscheinlichkeit $P(dist(C_\Gamma^{(n)}) \geq d)$ berechnen, dass der durch die Hinzunahme der zufällig gewählten Nichtnullspalten generierte Code eine Minimaldistanz von d oder höher besitzt. Das ist so möglich:

Zu $v \in \mathbb{F}_q^{k^*}$ und $j \leq n-m$ sei das Ereignis $E(v,j)$ wie folgt definiert:

$$E(v,j) := \text{„v hat mit mindestens } j \text{ der Spalten } \gamma_i \text{ Skalarprodukt ungleich Null“.}$$

Für *paarweise* linear abhängige Vektoren $v_1, v_2, ..., v_i \in \mathbb{F}_q^{k^*}$ und $j \leq n-m$ sind die Ereignisse $E(v_1,j), E(v_2,j), ..., E(v_i,j)$ offensichtlich total abhängig, da entweder jedes oder keines eintrifft:

$$E(v_1,j) \vee E(v_2,j) ... \vee E(v_i,j) \Leftrightarrow E(v_1,j) \wedge E(v_2,j) ... \wedge E(v_i,j)\,.$$

Setzen wir zu $\langle v \rangle \in PG_{k-1}(q)$ und $j \leq n-m$

$$E(\langle v \rangle, j) := \bigwedge_{u \in \langle v \rangle} E(u,j)$$

so gilt deshalb:

$$P(E(\langle v \rangle, j)) = P(E(v,j))\,.$$

Unter der (nicht ganz richtigen) Annahme, dass für verschiedene $\langle v \rangle \in PG_{k-1}(q)$ und $j_{\langle v \rangle} \leq n-m$ die jeweiligen Ereignisse $E(\langle v \rangle, j_{\langle v \rangle})$ stochastisch unabhängig sind, ist die gesuchte Wahrscheinlichkeit dann (mit $M := \{\langle v \rangle \in PG_{k-1}(q) : gew(v\Gamma) < d\}$):

$$P(dist(C_\Gamma^{(n)}) \geq d) \approx \prod_{\langle v \rangle \in M} P(E(\langle v \rangle, d - gew(v\Gamma))) = \prod_{\langle v \rangle \in M} s(n-m, d-gew(v\Gamma))\,.$$

Da in diesem Ausdruck der Faktor zu einem $\langle v \rangle \in M$ nur von $gew(v\Gamma)$ abhängt, lässt sich die Formel noch etwas reduzieren, wenn die Gewichtsverteilung

$$A_\Gamma := [l,u]\,(A_l, A_{l+1}, ..., A_{u-1}, A_u)$$

des von Γ erzeugten Codes bekannt ist:

$$\prod_{\langle v\rangle \in M} s(n-m, d-gew(v\Gamma)) = prob(A_\Gamma) := \prod_{i=l}^{min\{u,d-1\}} s(n-m, d-i)^{\frac{A_i}{q-1}} .$$

Wir wollen diese Wahrscheinlichkeit als *Bewertung* für Γ heranziehen:

$$eval(\Gamma) := prob(A_\Gamma) .$$

Für zwei verschiedene Matrizen Γ_1, Γ_2 kann $>$ nun einfach so definiert werden:

$$\Gamma_1 > \Gamma_2 :\Leftrightarrow eval(\Gamma_1) > eval(\Gamma_2) .$$

Einige Anmerkungen:

- Dass die Ereignisse $E(\langle v\rangle, j_{\langle v\rangle})$ für verschiedene $\langle v\rangle$ nicht stochastisch unabhängig sind, kann man sich leicht an einem Beispiel klarmachen: Ist etwa $m = n-1$ (d.h. es wird nur noch eine einzige Spalte γ zu Γ hinzugenommen), so gibt es genau q^{k-1} verschiedene Elemente aus $PG_{k-1}(q)$, deren Repräsentanten mit γ ein von Null verschiedenes Skalarprodukt haben. Für $q^{k-1}+1$ paarweise verschiedene $\langle v_1\rangle, \langle v_2\rangle, ..., \langle v_{q^{k-1}+1}\rangle \in PG_{k-1}(q)$ ist somit:

  $$P(\bigwedge_{i=1}^{q^{k-1}+1} E(\langle v_i\rangle, 1)) = 0 ,$$

  während

  $$\prod_{i=1}^{q^{k-1}+1} P(E(\langle v_i\rangle, 1)) \neq 0$$

  gilt, was die Anforderungen für stochastische Unabhängigkeit verletzt.
- Da sich die Approximation auf die Chance für *zufällige* Spaltenhinzunahme bezieht, besteht keinesfalls eine zwingende Korrespondenz des Wertes mit der „Chance“ bei *gezielter* Spaltenhinzunahme (im „objektiven“ Sinne ist es ohnehin missverständlich, hier von Wahrscheinlichkeiten zu reden - entweder ist die Erweiterung von Γ zur Generatormatrix eines $(n, k, \geq d, q)$-Codes möglich oder nicht).
- Dennoch ist dieses Kriterium m.E. besser als das aus 3.1: Auch hier wird grundsätzlich in jedem Schritt die Minimaldistanz erhöht (denn die Codeworte mit dem kleinsten Gewicht tragen auch den kleinsten Faktor zur Wahrscheinlichkeit bei), aber nicht um jeden Preis, denn die Anzahl der Codeworte eines jeden Gewichts unterhalb von d geht in die Formel mit ein.

- Für das oben genannte Beispiel würde auch mit diesem Kriterium noch der Paritätscheck durchgeführt: Für die Gewichtsverteilung $[2-5]\,(10,0,5)$ ist die Wahrscheinlichkeit zu den Parametern $n = 8$, $k = 4$, $q = 2$, $d = 4$ nämlich:

$$prob([2-5]\,(10,0,5)) = (\frac{448}{1125} + \frac{512}{3375})^{10} \approx 2.53 \cdot 10^{-3}\,,$$

bei Hinzunahme einer beliebigen Spalte von Gewicht drei zu I_4 ergibt sich dagegen die Gewichtsverteilung $[1-5]\,(1,6,6,1,1)$ und damit die Wahrscheinlichkeit

$$\begin{aligned} prob([1-5]\,(1,6,6,1,1)) &= (\frac{512}{3375})^1 \cdot (\frac{448}{1125} + \frac{512}{3375})^6 \cdot (\frac{392}{1125} + \frac{448}{1125} + \frac{512}{3375})^6 \\ &\approx 2.21 \cdot 10^{-3}\,. \end{aligned}$$

Für das Anfügen anderer Spalten ist die Bewertung jeweils kleiner als $2.21 \cdot 10^{-3}$.

Aber die Paritätserweiterung wird nun auch gelegentlich nicht mehr angewandt: Für $n = 9$, $k = 5$, $d = 4$, $q = 2$ etwa liefert der Paritätscheck nach Vorgabe der Einheitsmatrix I_5 die Gewichtsverteilung $[2-6]\,(15,0,15,0,1)$ und hierfür ist

$$prob([2-6]\,(15,0,15,0,1)) \approx 6.20 \cdot 10^{-5}\,.$$

Das Anhängen einer beliebigen Spalte von Gewicht vier führt dagegen zur Verteilung $[1-5]\,(1,10,10,5,5)$ mit

$$prob([1-5]\,(1,10,10,5,5)) \approx 6.47 \cdot 10^{-5}\,.$$

Hier würde der Paritätscheck also verworfen und stattdessen die erste untersuchte Spalte von Gewicht vier gewählt (die Werte für das Ergänzen von Spalten mit Gewicht drei und weniger sind wieder kleiner als die beiden angeführten).

Kapitel 4

Erweiterungen des Basisalgorithmus

Wir wollen nun im Folgenden, sofern nicht anders angegeben, davon ausgehen, dass der Vergleichsoperator $>$ wie in 3.2 implementiert ist. Ebenso habe, für eine Generatormatrix Γ, $eval(\Gamma)$ die dort angeführte Bedeutung. Jetzt soll der in 2.2 eingeführte Basisalgorithmus schrittweise weiter ausgebaut werden:

4.1 Vertiefte Suche

In 3.2 wurde ausgeführt, dass Algorithmus 1 für den Fall $n = 8$, $k = 4$, $d = 4$ und $q = 2$ bei Vorgabe der Einheitsmatrix I_4 den Paritätscheck wählen würde. Weiter wurde gezeigt, dass danach die Konstruktion eines (8,4,4,2)-Codes nicht mehr möglich ist. Würde der Algorithmus allerdings einen Schritt weiter „denken“, so könnte diese Sackgasse gemieden werden:

Die nach dem Paritätscheck entstehende Matrix

$$\Gamma_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

generiert einen Code mit Gewichtsverteilung $[2-5]\,(10, 0, 5)$ und $eval(\Gamma_1) \approx 2.53 \cdot 10^{-3}$. Die Hauptalternative zum Paritätscheck ist das Anfügen einer Spalte von Gewicht drei, was eine Matrix des Typs

$$\Gamma_2 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

ergibt, mit der Verteilung $[1-5]\,(1,6,6,1,1)$ und $eval(\Gamma_2) \approx 2.21 \cdot 10^{-3}$. Γ_1 wird im nächsten Schritt erweitert zu

$$\Gamma_1' = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Das liefert die Gewichtsverteilung $[2-5]\,(4,6,3,2)$ und $eval(\Gamma_1') \approx 1.50 \cdot 10^{-3}$. Aus Γ_2 dagegen entsteht analog

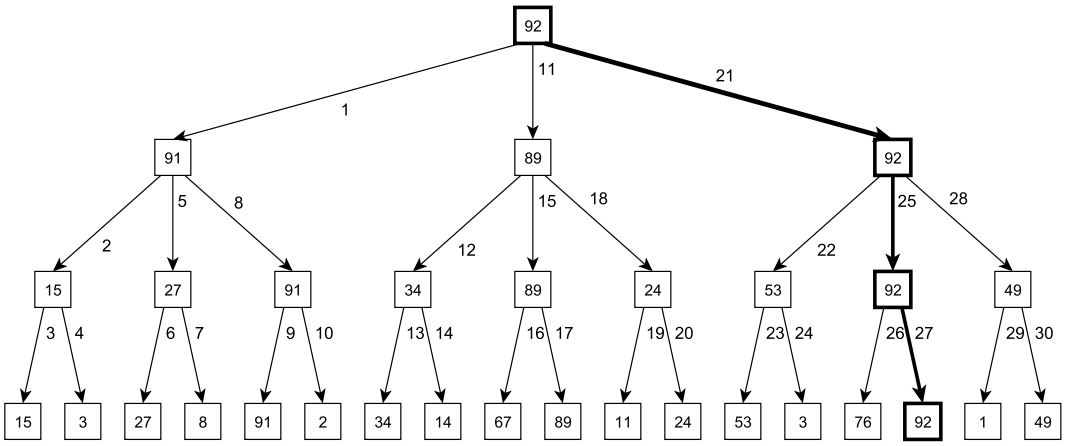
$$\Gamma_2' = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

mit Verteilung $[2-6]\,(3,8,3,0,1)$ und $eval(\Gamma_2') \approx 3.23 \cdot 10^{-3}$.

Der Algorithmus würde also im zweiten Schritt seine zuvor getroffene Wahl „bereuen“, denn die Lage, die sich aus Γ_2 entwickelt hat, ist vielversprechender als jene, die sich aus Γ_1 ergab. Wäre es also nicht sinnvoll gewesen, wenn er vor der Wahl der ersten Spalte statt einem gleich zwei oder mehr Schritte vorausgedacht hätte? Es klingt natürlich einleuchtend, dass „weit rechnen“ besser ist als „kurz rechnen“. Aber ganz so klar ist die Sache nicht: Schließlich könnte es im Beispiel ja so sein, dass sich die Verhältnisse nach dem dritten Schritt wieder umdrehen, also die beste Erweiterung von Γ_1' einen besseren Wert liefert als die beste von Γ_2'. Ein Zusammenhang der Art „eine tiefere Suche liefert bessere Ergebnisse“ wäre dann nicht direkt gegeben. Nun, um noch einmal auf den in 2.2 angestellten Vergleich mit Schachprogrammen zurückzukommen: Die Erfahrung zeigt deutlich, dass ein Schachprogramm mit zunehmender Suchtiefe spielstärker wird. Eine (von vermutlich mehreren, sich ergänzenden) Erklärungen dürfte sein, dass die taktischen (= kurzfristigen) Verwicklungen in einer Schachstellung in der Regel begrenzt sind und sich ab einer ausreichend hohen Suchtiefe die Ergebnisse für die zur Wahl stehenden Züge stabilisieren. Kann z. B. in einer Stellung die am Zug befindliche Partei durch eine zwingende, zehnzügige Kombination unter sonst ausgeglichenen Verhältnissen eine Figur gewinnen (ein im Schach mit fast an Sicherheit grenzender Wahrscheinlichkeit zum Sieg ausreichender Vorteil) und kann diese Kombination nur durch einen einzigen Zug eingeleitet werden, so können bei den Suchen mit Tiefen von ein bis neun Zügen zwar durchaus noch andere Möglichkeiten bevorzugt werden; ab Suchtiefe zehn allerdings wird auf jeden Fall der Gewinnzug gefunden und auch bei höheren Tiefen beibehalten werden. Für weniger klare Fälle (relativ selten im Schach kann man gleich zwingend eine Figur gewinnen) könnte man wie folgt verallgemeinern: Zwischen einem guten Zug **A** und einem weniger guten Zug **B** wird es normalerweise einen Unterschied geben, der nach einer gewissen Anzahl **z** von Zügen in der Art zum Tragen kommt, dass ihn die Bewertungsfunktion des Schachprogramms erfassen kann. Ab einer Suchtiefe von **z** Zügen kann das Programm dann „sicher“ erkennen, dass **A** besser ist als **B**, während die Ergebnisse für kleinere Tiefen noch hin- und herpendeln können.

Überträgt man diese Überlegung auf die Suche nach linearen Codes, so ergibt sich folgender Schluss: Unter der Annahme, dass sich in den *meisten Fällen* der Unterschied zwischen einer „günstigen“ und einer „ungünstigen“ Spaltenwahl dergestalt äußert, dass ab einer bestimmten Zahl von Folgeschritten die Zwischenergebnisse („$eval(\Gamma)$“) für die günstige *konstant* besser sind als für die ungünstige, erhöht sich mit einer tieferen Suche die Chance, diese kritische Schwelle zu erreichen und die beiden Spalten „korrekt“ miteinander zu vergleichen. Diese Annahme erscheint zumindest plausibel und nach dem subjektiven Eindruck des Autors liefert eine tiefere Suche nach dem im Folgenden beschriebenen Schema bei sonst gleichen Bedingungen in der Mehrzahl der Fälle bessere Ergebnisse als die Einschrittsuche aus Algorithmus 1. Allerdings bleibt die Frage, ob der für die tiefere Suche erforderliche Rechenaufwand nicht an anderer Stelle besser investiert wäre; in Verbindung mit dem später erläuterten Backtracking-Verfahren etwa erlaubt eine „seichte“ Suche einen wesentlich höheren Durchsatz an besuchten Knoten pro Zeiteinheit, was vielleicht sogar mehr als nur eine ausreichende Kompensation für die etwas geringere „Suchqualität“ darstellt. Ein einfacher Entwurf für eine Suche mit Tiefe t, der im Wesentlichen die aus der Informatik bekannte vollständige Tiefensuche einsetzt, findet sich in Algorithmus 2.

Der Aufbau der Funktion *main* gleicht im Wesentlichen Algorithmus 1; statt jedoch die Matrizen $(\Gamma \,|\, v^t)$ und $(\Gamma \,|\, \gamma)$ direkt durch $>$ zu vergleichen, werden stattdessen die Rückgabewerte der Funktion *depthFirstSearch* herangezogen: Die Variable *bestEval* speichert die Bewertung zur bisher besten gefundenen Spalte γ, *tempEval* den Wert für das aktuelle v^t. Gilt $tempEval > bestEval$, so wird $bestEval = tempEval$ gesetzt und v^t zum neuen Favoriten (Z. 7-13). Die Initialisierung von *bestEval* in Zeile 6 mit dem Wert -1.0 bewirkt, dass für das erste durchlaufene v diese Bedingung auf jeden Fall erfüllt ist.
Die Funktion *depthFirstSearch* enthält die eigentliche Tiefensuche: Ihr erstes Argument Γ ist die zu bewertende Matrix, das zweite, *depth*, die verbleibende Resttiefe. Ist die Bedingung $\Gamma \in \mathbb{F}_q^{k \times n}$ aus Zeile 23, die verhindern soll, dass der Algorithmus Matrizen mit mehr als n Spalten betrachtet, im gesamten rekursiven Ablauf für $depth > 0$ niemals erfüllt (d. h. das beim einleitenden Aufruf übergebene Γ bestand aus höchstens $n - depth$ Spalten), so entspricht der Rückgabewert von *depthFirstSearch* dem maximalen Wert $eval(\overline{\Gamma})$ über alle Matrizen $\overline{\Gamma}$, die aus Γ durch Anfügen von *depth* Spalten entstehen. Dies macht man sich leicht durch Induktion nach *depth* klar: Ist $depth = 0$, so wird einfach die Bewertung $eval(\Gamma)$ nach 3.2 zurückgegeben, was offensichtlich die Induktionsbasis liefert. Andernfalls jedoch wird, nach demselben Prinzip wie in *main*, diejenige Spalte v^t ermittelt, bei welcher der (rekursive!) Aufruf von *depthFirstSearch* mit den Argumenten $(\Gamma \,|\, v^t)$ und $depth - 1$ den höchsten Wert zurückgibt. Zusammen mit der Induktionsvoraussetzung ergibt sich damit unmittelbar der Induktionsschluß.
Wie für Algorithmus 1 bleibt anzumerken, dass die im m-ten Schritt in *main* getroffene Spaltenwahl im Falle mehrerer, hinsichtlich des Rückgabewerts von *depthFirstSearch* gleichwertiger Kandidaten, davon abhängt, welche der Spalten in der Schleife ab Zeile 7 zuerst untersucht wird. Der Rückgabewert von *depthFirstSearch* selbst dagegen ist unabhängig von der für die *for*-Schleife in Zeile 27 gewählten Reihenfolge.



Schematischer Ablauf von *depthFirstSearch* aus Algorithmus 2: Die Kantennumerierung gibt die Durchlaufreihenfolge an, die Zahlen in den Blattknoten entsprechen (proportional) den Rückgabewerten von *eval*. Die Werte im Inneren ergeben sich durch (rekursives) Maximieren über die Kinder des jeweiligen Knotens.

Eingabe : **int** n, k, d, q: Parameter des linearen Codes
int t: Suchtiefe
MATRIX $\Gamma_0 \in \mathbb{F}_q^{k \times n_0}$: vorgegebene Ausgangsmatrix ($n_0 \leq n$)
Ausgabe : Generatormatrix Γ zu $(n, k, \geq d, q)$-Code oder FAILED

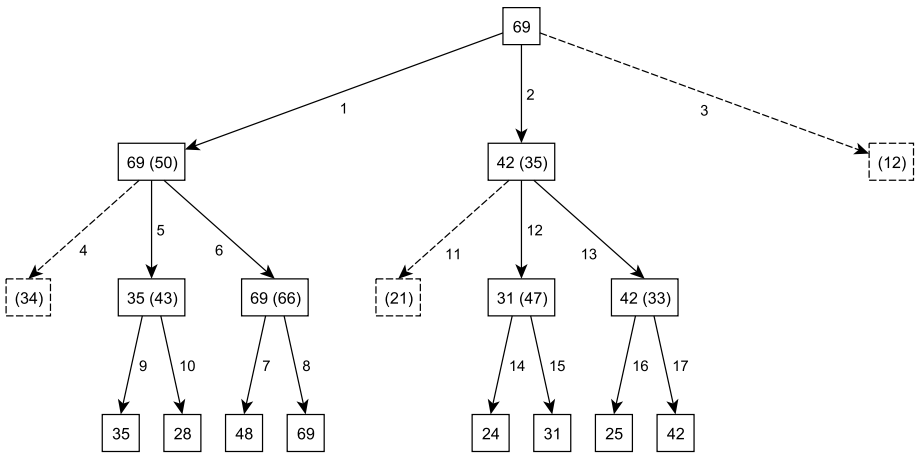
```
1  PROCEDURE main() : MATRIX bzw. FAILED
2  {
3    MATRIX Γ ← Γ_0;
4    for int m from n_0 to n − 1 do
5      COLUMN γ;
6      real bestEval ← −1.0;
7      foreach ⟨v⟩ ∈ PG_{k−1}(q) do
8        real tempEval ← depthFirstSearch((Γ | v^t), t − 1);
9        if tempEval > bestEval then
10         γ ← v^t;
11         bestEval ← tempEval;
12       endif
13     endfch
14     Γ ← (Γ | γ);
15   endfor
16   if dist(LinearCode(Γ)) ≥ d then
17     return Γ;
18   endif
19   return FAILED;
20 }
21 PROCEDURE depthFirstSearch(MATRIX Γ, int depth) : real
22 {
23   if depth = 0 or Γ ∈ F_q^{k×n} then
24     return eval(Γ);
25   endif
26   real bestEval ← −1.0;
27   foreach ⟨v⟩ ∈ PG_{k−1}(q) do
28     real tempEval ← depthFirstSearch((Γ | v^t), depth − 1);
29     if tempEval > bestEval then
30       bestEval ← tempEval;
31     endif
32   endfch
33   return bestEval;
34 }
```

Algorithmus 2 : vollständige Tiefensuche

4.2 Selektive Suche

Ein großes Problem von Algorithmus 2 ist allerdings der mit der Tiefe drastisch zunehmende Rechenaufwand. Mit $s := |PG_{k-1}(q)| = \frac{q^k-1}{q-1}$, beträgt er offensichtlich $\mathcal{O}(s^t)$, wodurch oft selbst für relativ kleine t die praktische Durchführung aus Zeitgründen unmöglich wird (daran ändert auch das Hinzufügen von durchaus vorhandenen Verbesserungen zu Algorithmus 2 nichts Wesentliches). Um dennoch Suchen mit höheren Tiefen zu erlauben, kann man den Verzweigungsfaktor „künstlich“ beschränken: Statt in einem Knoten des Suchbaumes die nachfolgenden Schritte für jede mögliche Spaltenwahl durchzuführen, wird zunächst eine Vorauswahl der Spalten durch eine Suche mit niedriger Tiefe bestimmt und nur für diese anschließend die tiefere Suche durchgeführt; alle anderen Spalten bleiben dagegen innerhalb dieses Knotens unberücksichtigt. Die dahinterstehende Annahme ist, dass mit hoher Wahrscheinlichkeit nur die vorausgewählten Spalten für die tiefe Suche relevant sind und man deshalb im Idealfall dieselben Ergebnisse erhielte wie bei einer vollständigen Suche mit derselben Tiefe. Das damit verbundene Risiko ist klar: Eine Garantie, dass man durch die Vorselektion nicht auch gelegentlich eine (hinsichtlich der tieferen Suche) aussichtsreiche Spalte aussortiert, gibt es nicht. Algorithmus 3 zeigt eine mögliche Implementierung dieser Idee, wenn für die Vorauswahl der Spalten eine Suche mit Tiefe eins eingesetzt wird; um die Formulierung des im nächsten Kapitel vorgestellten Backtracking zu erleichtern, wurde die Routine selSearch() bereits so implementiert, dass sie eine (nach der zweiten Komponente absteigend) sortierte Liste aus Paaren von Spalten mit Ihren in der Suche ermittelten Bewertungen zurückliefert.

In *main* wird in jedem Schritt diese Liste für das aktuelle Γ mit der vorgegebenen Tiefe t ermittelt (Z. 5) und die zum ersten Listeneintrag gehörige Spalte an Γ angehängt (Z. $6-7$); der Rest ist selbsterklärend. Etwas mehr passiert in *selSearch*: In den Zeilen $16-19$ wird zunächst, für jedes $v \in PG_{k-1}(q)$, v^t zusammen mit der Bewertung für die Matrix $(\Gamma \,|\, v^t)$ in der Liste *selList* gespeichert. In Zeile 20 werden die Einträge aus *selList* absteigend nach Bewertung sortiert. Das erste Tupel entspricht nun genau dem Paar $(\gamma, bestEval)$, welches Algorithmus 2 nach Verlassen der *for*-Schleife in Zeile 13 bei Vorliegen der Matrix Γ für die Suchtiefe $t = 1$ ermittelt hätte. Hat das zweite Argument r, welches die verbleibende Resttiefe anzeigt, den Wert 1 (bzw. besteht das übergebene Γ bereits aus $n-1$ Spalten), so muss keine tiefere Suche mehr erfolgen und die Liste kann einfach zurückgegeben werden. Andernfalls setzt nun der Selektionsmechanismus ein: Zeile 24 reduziert *selList* auf die ersten b Einträge. Nur für diese wird nun die vertiefte Suche durchgeführt: Dazu wird, für jeden Eintrag (*col*, *eval*) aus *selList*, die Bewertungskomponente *eval* mit dem „Rückgabewert“ von *selSearch* zur Tiefe $r-1$ für die Matrix $(\Gamma \,|\, col)$ überschrieben (genauer gesagt, mit der zweiten Komponente des obersten Tupels aus der zurückgegebenen Liste)(Z. $25-28$). Anschließend wir die Liste, nach erneuter Sortierung, zurückgegeben. Die Reihenfolge, in welcher $PG_{k-1}(q)$ in Zeile 17 durchlaufen wird, hat hier unter Umständen durchaus wesentlichen Einfluss auf das Suchresultat: Nehmen wir z. B. an, wir hätten zwei Tupel (μ, e_μ) und (ν, e_ν) mit gleicher zweiter Komponente, d. h. es gilt $e_\mu = e_\nu$. Es sind nun Fälle denkbar, in denen beim Selektionsschritt in Zeile 24, je nach Sortierung, entweder (μ, e_μ) oder (ν, e_ν) aus der Liste entfernt wird. Da keinesfalls klar ist, dass sich diese beiden Tupel hinsichtlich einer tieferen Suche gleich „verhalten“, können die Ergebnisse in diesen beiden Fällen durchaus unterschiedlich ausfallen. Weil es sich hier jedoch um „Zufälligkeiten“ handelt, können in qualitativer Hinsicht alle Durchlaufreihenfolgen, bei denen keine weiteren Informationen über die $\langle v \rangle \in PG_{k-1}(q)$ verwendet werden, als gleichwertig angesehen werden.



Schematischer Ablauf von *selSearch* aus Algorithmus 3 für $b = 2$: In Klammern die zur Vorselektion herangezogenen Werte. Der diesbezüglich besser bewertete Knoten wird hier im Anschluß zuerst untersucht.

Eingabe : **int** n, k, d, q: Parameter des linearen Codes
int t: Suchtiefe
int b: Zahl der Spalten pro Knoten, für die eine vertiefte Suche erfolgt
MATRIX $\Gamma_0 \in \mathbb{F}_q^{k \times n_0}$: vorgegebene Ausgangsmatrix ($n_0 \leq n$)
Ausgabe : Generatormatrix Γ zu $(n, k, \geq d, q)$-Code oder FAILED

Hilfsroutinen :
getFirstElement(**LIST** $\langle$**T**$\rangle$ L): gib erstes Element von L zurück
pushBack(**LIST** $\langle$**T**$\rangle$ L, **T** x): füge Element x zu L hinzu
orderByEval(**LIST** $\langle$**COLUMN** $\times$ **real**$\rangle$ L): ordne L fallend in 2. Komponente
reduceList(**LIST** $\langle$**T**$\rangle$ L, **int** s): reduziere L auf die ersten s Elemente

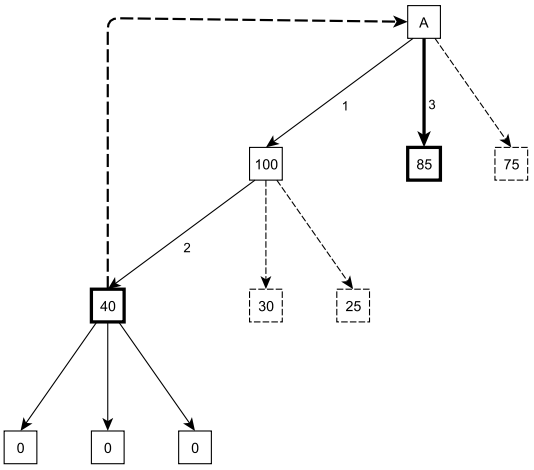
```
1  PROCEDURE main() : MATRIX bzw. FAILED
2  {
3    MATRIX Γ ← Γ_0;
4    for int m from n_0 to n − 1 do
5      LIST ⟨COLUMN × real⟩ tempList ← selSearch(Γ, t);
6      (COLUMN γ, real dummy) ← getFirstElement(tempList);
7      Γ ← (Γ | γ);
8    endfor
9    if dist(LinearCode(Γ)) ≥ d then
10     return Γ;
11   endif
12   return FAILED;
13 }
14 PROCEDURE selSearch(MATRIX Γ, int r) : LIST ⟨COLUMN × real⟩
15 {
16   LIST ⟨COLUMN × real⟩ selList;
17   foreach ⟨v⟩ ∈ PG_{k−1}(q) do
18     pushBack(selList, (v^t, eval((Γ | v^t))));
19   endfch
20   orderByEval(selList);
21   if r = 1 or Γ ∈ F_q^{k×(n−1)} then
22     return selList;
23   endif
24   reduceList(selList, b);
25   foreach (col, eval) ∈ selList do
26     LIST ⟨COLUMN × real⟩ tempList ← selSearch((Γ | col), r − 1);
27     (COLUMN dummy, eval) ← getFirstElement(tempList);
28   endfch
29   orderByEval(selList);
30   return selList;
31 }
```

Algorithmus 3 : selektive Tiefensuche

4.3 Backtracking

In den bisher vorgestellten Algorithmen wird stets nur ein einziger „Anlauf“ bei dem Versuch unternommen, einen (n, k, d, q)-Code zu konstruieren: Unmittelbar nach Hinzunahme der n-ten Spalte endet die Suche, unabhängig davon, ob sie erfolgreich verlief oder nicht. Das ist im Falle eines Scheiterns natürlich unbefriedigend, viel besser wäre es, wenn der Algorithmus dann durch Rücknahme einiger der zuletzt gewählten Spalten selbstständig an eine Stelle zurückkehrt, an der es „vielversprechende Alternativen“ zu der ursprünglich festgelegten Spalte gab. Diese sollten dann nacheinander in der gleichen Weise wie der Hauptkandidat untersucht werden. Bringt auch das keinen Erfolg, kann man erneut einige Schritte zurückgehen, bis zum nächsten Punkt, an dem aussichtsreiche Varianten existieren. Erst wenn es keinen solchen Punkt mehr gibt, wird die Suche gestoppt. Die Frage, die sich dabei stellt: was will man unter einer „vielversprechenden Alternative“ verstehen? Es bietet sich an, als Kriterium hierfür die Werte heranzuziehen, die die Suchfunktion (etwa selSearch(...) in Algorithmus 3) zurückgibt. Liegt der Wert w_2 für eine Spalte γ_2 nur „knapp“ unterhalb des Wertes w_1 für die beste Spalte γ_1, so wird diese als Alternativmöglichkeit zu γ_1 betrachtet; „knapp“ kann wiederum durch eine feste Grenze $0 \leq a \leq 1$ definiert werden, die das Verhältnis $\frac{w_2}{w_1}$ übersteigen muss. Algorithmus 4 kombiniert diesen Ansatz mit der selektiven Suche aus Algorithmus 3; die Funktion selSearch(...) soll genau die gleiche Gestalt wie dort besitzen und wird deshalb nicht erneut aufgeführt.

Für jeden Schritt m wird die von *selSearch* zurückgegebene Ergebnisliste in *resList[m]* abgelegt (Z. 10). Daraus wird das erste Element entfernt und die zu diesem Tupel gehörige Spalte in der Variable γ, sowie ihre Bewertung in *bestEval[m]* gespeichert (Z. 11). γ wird dann an Γ angefügt und dies solange iteriert, bis Γ n Spalten besitzt. Nun wird die *while*-Schleife (Zeilen $9-14$) verlassen und geprüft, ob eine Minimaldistanz von mindestens d erreicht wurde. Falls nicht, so wird die letzte Spalte von Γ entfernt und m dekrementiert. Ist *resList[m]* nicht leer, entnehmen wir das oberste Tupel (μ, e_μ); erfüllt es die Bedingung $e_\mu \geq a \cdot bestEval[m]$, so wird die Spalte μ an Γ angehängt, m inkrementiert und anschließend die Suche in Zeile 9 in gewohnter Weise wieder aufgenommen (Z. $22-30$). Andernfalls werden solange weitere Spalten von Γ entfernt und m entsprechend verringert, bis die Bedingung für den m-ten Schritt erfüllt ist; danach geht es wieder in Zeile 9 weiter (*while*-Schleife von Zeile $19-31$). Die alten Ergebnislisten (sowie *bestEval[...]*) werden dann für die nachfolgenden Schritte automatisch in der Schleife von Zeile $9-14$ mit den aktuell gültigen überschrieben. Ist allerdings auch noch nach Rücknahme der ersten „selbstgewählten“ Spalte (d. h. für $\Gamma = \Gamma_0$) die Bedingung verletzt, so gibt es keine weiteren Alternativen. Die Suche ist dann gescheitert und ein entsprechender Wert wird zurückgegeben.



Schematischer Ablauf von Algorithmus 4 für $a = 0.8$: Die Zahlen in den Kästchen einer Tiefenstufe repräsentieren die Bewertungskomponenten der Tupel aus der jeweils von *selSearch* zurückgegebenen Liste. Knoten zu gestrichtelten Vorwärtskanten stellen (bei diesem a) keine Alternative dar. Erkennt der Algorithmus im Knoten, auf den Kante 2 zeigt, dass keine erfolgreiche Erweiterung möglich ist, kehrt er zum Knoten A zurück und wählt Kante 3 als Alternative zu Kante 1.

Eingabe : **int** n, k, d, q, t, b: wie in Algorithmus 3
real a: Mindestwert für $\frac{w_2}{w_1}$ zur Festlegung von Alternativen
MATRIX $\Gamma_0 \in \mathbb{F}_q^{k \times n_0}$: vorgegebene Ausgangsmatrix ($n_0 \leq n$)

Ausgabe : Generatormatrix Γ zu $(n, k, \geq d, q)$-Code oder FAILED

Hilfsroutinen :
popFirstElement(**LIST**⟨**T**⟩ L): entferne erstes Element aus L, gib es zurück
notEmpty(**LIST**⟨**T**⟩ L): gib true zurück genau dann, wenn L nicht leer
removeLastColumn(**MATRIX** Γ): streiche letzte Spalte aus Γ
selSearch(**MATRIX**, **int**): vgl. Algorithmus 3

```
1  PROCEDURE main() : MATRIX bzw. FAILED
2  {
3    LIST ⟨COLUMN × real⟩ resList[n];
4    real bestEval[n];
5    MATRIX Γ ← Γ_0;
6    int m ← n_0;
7    bool foundAlternative ← true;
8    while foundAlternative do
9      while m ≤ n − 1 do
10       resList[m] ← selSearch(Γ, t);
11       (COLUMN γ, bestEval[m]) ← popFirstElement(resList[m]);
12       Γ ← (Γ | γ);
13       m ← m + 1;
14     endw
15     if dist(LinearCode(Γ)) ≥ d then
16       return Γ;
17     endif
18     foundAlternative ← false;
19     while m > n_0 do
20       removeLastColumn(Γ);
21       m ← m − 1;
22       if notEmpty(resList[m]) then
23         (COLUMN γ, real tempEval) ← popFirstElement(resList[m]);
24         if tempEval ≥ a · bestEval[m] then
25           Γ ← (Γ | γ);
26           m ← m + 1;
27           foundAlternative ← true;
28           break;
29         endif
30       endif
31     endw
32   endw
33   return FAILED;
34 }
```

Algorithmus 4 : Backtracking + selektive Suche

4.4 Isometrieerkennung

Wie schon in Abschnitt 1.6 ausgeführt, gibt es zu demselben linearen Code C normalerweise sehr viele verschiedene Generatormatrizen, die C erzeugen. Außerdem existieren auch andere Codes, die zwar von C formal verschieden sind, jedoch in den *wesentlichen* Eigenschaften übereinstimmen - ein Beispiel sind die zu C linear isometrischen Codes. In gleicher Weise können dann natürlich auch die zugehörigen Generatormatrizen als äquivalent betrachtet werden. Wir wollen jetzt untersuchen, wann zwei Generatormatrizen $\Gamma_1 \neq \Gamma_2$ im Sinne der hier behandelten Algorithmen als gleich anzusehen sind. Grob formuliert sollte dies genau dann der Fall sein, wenn für jede Stelle im Algorithmus, an der die Matrix Γ_1 auftritt, diese durch Γ_2 ersetzt könnte, ohne dass sich der Programmablauf und die Resultate *wesentlich* verändern. Der Vorteil, den die Erkennung einer solchen Äquivalenz mit sich bringt, liegt auf der Hand: Tritt an einer Stelle im Programm Γ_2 auf und wurde die gleiche Situation zuvor schon mit Γ_1 anstelle von Γ_2 untersucht, so können die dabei ermittelten Ergebnisse, sofern sie in geeigneter Weise gespeichert wurden, risikolos übernommen werden, ohne alles noch einmal berechnen zu müssen.

Bei allen in dieser Arbeit entwickelten Algorithmen ist eine Generatormatrix Γ ausreichend durch die Gewichtsverteilungen der Codes charakterisiert, die von Γ selbst bzw. durch eine Erweiterungsmatrix von Γ erzeugt werden, also ein $\tilde{\Gamma}$ vom Typ

$$\tilde{\Gamma} = (\Gamma \,|\, \gamma_1^t \,|\, \gamma_2^t \,|\, ... \,|\, \gamma_i^t),\ i \in \mathbb{N}^* \,.$$

Das motiviert die folgende Definition:

Definition 4.1 *Für eine Matrix $\Gamma \in \mathbb{F}_q^{k\times m}$ sei $wd(\Gamma)$ die Gewichtsverteilung der von Γ generierten Worte, d. h. $wd(\Gamma) = (A_i)_{i=0..m}$ mit $A_i = |\{v \in \mathbb{F}_q^k : gew(v\Gamma) = i\}|$. Zwei Matrizen $\Gamma_1, \Gamma_2 \in \mathbb{F}_q^{k\times m}$ nennen wir* erweiterungsäquivalent*, in Notation $\Gamma_1 \sim_E \Gamma_2$, wenn gilt:*

Für $i \in \mathbb{N}_0$ und beliebige $\gamma_1, \gamma_2, ..., \gamma_i \in \mathbb{F}_q^k$ gibt es stets $\overline{\gamma}_1, \overline{\gamma}_2, ..., \overline{\gamma}_i \in \mathbb{F}_q^k$, s. d.

$$wd((\Gamma_1 \,|\, \gamma_1^t \,|\, \gamma_2^t \,|\, ... \,|\, \gamma_i^t)) = wd((\Gamma_2 \,|\, \overline{\gamma}_1^t \,|\, \overline{\gamma}_2^t \,|\, ... \,|\, \overline{\gamma}_i^t))$$

und umgekehrt:

Für $i \in \mathbb{N}_0$ und beliebige $\overline{\gamma}_1, \overline{\gamma}_2, ..., \overline{\gamma}_i \in \mathbb{F}_q^k$ gibt es jeweils $\gamma_1, \gamma_2, ..., \gamma_i \in \mathbb{F}_q^k$, s. d.

$$wd((\Gamma_2 \,|\, \overline{\gamma}_1^t \,|\, \overline{\gamma}_2^t \,|\, ... \,|\, \overline{\gamma}_i^t)) = wd((\Gamma_1 \,|\, \gamma_1^t \,|\, \gamma_2^t \,|\, ... \,|\, \gamma_i^t)) \,.$$

Offensichtlich gilt:

Lemma 4.1 *Für feste Parameter m, k, q ist die Relation $\sim_E$ eine Äquivalenzrelation auf der Menge $\{\Gamma \in \mathbb{F}_q^{k\times m}\}$.*

Wie intuitiv zu erwarten, genügen Generatormatrizen zu linear isometrischen Codes der Bedingung aus Definition 4.1:

Satz 4.1 *Sind $\Gamma_1, \Gamma_2 \in \mathbb{F}_q^{k\times m}$ Generatormatrizen zu linear isometrischen Codes C_1 bzw. C_2, so sind Γ_1 und Γ_2 erweiterungsäquivalent.*

Beweis:
Da C_1 linear isometrisch zu C_2 ist, folgt nach Definition 1.1:

$$\exists M \in M_m(q) : C_1 M = C_2 \,.$$

Also erzeugen $\Gamma_1 M$ und Γ_2 denselben linearen Code C_2. Nach Satz 1.3 ist dies gleichbedeutend mit der Existenz einer invertierbaren Matrix $B \in \mathbb{F}_q^{k\times k}$, s. d. $\Gamma_2 = B\Gamma_1 M$.
Seien nun $\gamma_1, \gamma_2, ..., \gamma_i \in \mathbb{F}_q^k$ und $\overline{\Gamma}_1 := (\Gamma_1 \,|\, \gamma_1^t \,|\, \gamma_2^t \,|\, ... \,|\, \gamma_i^t)$.
Setze

$$\overline{M} := \begin{pmatrix} M & 0 \\ 0 & I_i \end{pmatrix}$$

und

$$\overline{\Gamma}_2 := B\overline{\Gamma}_1\overline{M} = (\Gamma_2 \,|\, B\gamma_1^t \,|\, B\gamma_2^t \,|\, ... \,|\, B\gamma_i^t) \,.$$

Da $\overline{M} \in M_{m+i}(q)$, sind die von $\overline{\Gamma}_1$ und $\overline{\Gamma}_2$ erzeugten Codes linear isometrisch, haben also insbesondere dieselbe Gewichtsverteilung. Somit bilden $\overline{\gamma}_1^t, \overline{\gamma}_2^t, ..., \overline{\gamma}_i^t$ mit $\overline{\gamma}_j^t := B\gamma_j^t$ die nach Definition 4.1 benötigten Erweiterungsspalten zu Γ_2, um einen Code mit derselben Gewichsverteilung wie $\overline{\Gamma}_1$ zu erhalten; umgekehrt liefert, zu vorgegebenen $\overline{\gamma}_1^t, \overline{\gamma}_2^t, ..., \overline{\gamma}_i^t$, die Setzung $\gamma_j^t := B^{-1}\overline{\gamma}_j^t$ die entsprechende Erweiterung von Γ_1.

□

Wir wollen nun zeigen, dass auch die Generatormatrizen zu semilinear isometrischen Codes erweiterungsäquivalent sind. Dazu benötigen wir noch etwas Vorarbeit:

Definition 4.2 *Ist α ein Körperautomorphismus auf $\mathbb{F}_q$, so sei*

- *für einen Vektor $v \in \mathbb{F}_q^k$ mit $v = (v_1, v_2, ..., v_k)$: $\alpha(v) := (\alpha(v_1), \alpha(v_2), ..., \alpha(v_k))$.*
- *für einen linearen Code C: $\alpha(C) := \{\alpha(c) : c \in C\}$*
- *für eine Generatormatrix $\Gamma \in \mathbb{F}_q^{k\times m}$ mit $\Gamma = (\gamma_{ij})$: $\alpha(\Gamma) := (\alpha(\gamma_{ij}))$.*

Lemma 4.2 *Sei $\Gamma \in \mathbb{F}_q^{k\times m}$ eine Generatormatrix, $\alpha \in Aut(\mathbb{F}_q)$. Dann gilt: $C_{\alpha(\Gamma)} = \alpha(C_\Gamma)$. Weiterhin haben C_Γ und $C_{\alpha(\Gamma)}$ dieselbe Gewichtsverteilung.*

Beweis:

$$\begin{aligned}
C_{\alpha(\Gamma)} &= \{v\alpha(\Gamma) : v \in \mathbb{F}_q^k\} \overset{v\mapsto\alpha(v)bij.}{=} \{\alpha(v)\alpha(\Gamma) : v \in \mathbb{F}_q^k\} = \\
&= \{(\sum_{i=1..k} \alpha(v_i)\alpha(\gamma_{i1}), ..., \sum_{i=1..k} \alpha(v_i)\alpha(\gamma_{im})) : (v_1, v_2, ..., v_k) \in \mathbb{F}_q^k\} = \\
&= \{(\alpha(\sum_{i=1..k} v_i\gamma_{i1}), ..., \alpha(\sum_{i=1..k} v_i\gamma_{im})) : (v_1, v_2, ..., v_k) \in \mathbb{F}_q^k\} = \\
&= \{\alpha(v\Gamma) : v \in \mathbb{F}_q^k\} = \alpha(C_\Gamma).
\end{aligned}$$

$C_{\alpha(\Gamma)}$ entsteht also durch komponentenweise Anwendung von α auf die Codeworte von C_Γ; da deren Gewicht dabei unverändert bleibt, folgt auch der zweite Teil der Behauptung.

□

Eine einfache Folgerung ist

Lemma 4.3 *Sei $\Gamma \in \mathbb{F}_q^{k\times m}$ eine Generatormatrix, $\alpha \in Aut(\mathbb{F}_q)$. Dann sind Γ und $\alpha(\Gamma)$ erweiterungsäquivalent.*

Beweis:
Zu $\gamma_1, \gamma_2, ..., \gamma_i \in \mathbb{F}_q^k$ setze $\overline{\gamma}_j := \alpha(\gamma_j)$. Dann erzeugen, nach Lemma 4.2, die Matrizen $(\Gamma \,|\, \gamma_1^t \,|\, \gamma_2^t \,|\, ... \,|\, \gamma_i^t)$ und $(\alpha(\Gamma) \,|\, \overline{\gamma}_1^t \,|\, \overline{\gamma}_2^t \,|\, ... \,|\, \overline{\gamma}_i^t)$ Codes mit derselben Gewichtsverteilung. Analog liefert, zu $\overline{\gamma}_1, \overline{\gamma}_2, ..., \overline{\gamma}_i \in \mathbb{F}_q^k$, die Setzung $\gamma_j := \alpha^{-1}(\overline{\gamma}_j)$ eine geeignete Erweiterung von Γ.

□

Nun können wir den gewünschten Satz herleiten:

Satz 4.2 *Generatormatrizen zu semilinear isometrischen Codes sind erweiterungsäquivalent.*

Beweis:
Seien $\Gamma_1, \Gamma_2 \in \mathbb{F}_q^{k\times m}$ Generatormatrizen zu semilinear isometrischen Codes C_{Γ_1} und C_{Γ_2}. Nach Definition 1.2 folgt daraus die Existenz einer semilinearen Isometrie $\sigma = (\alpha, \iota)$, mit einer linearen Isometrie ι und $\alpha \in Aut(\mathbb{F}_q)$, s. d. $C_{\Gamma_2} = \iota(\alpha(C_{\Gamma_1}))$. Wir setzen $\overline{C} := \alpha(C_{\Gamma_1})$ und $\overline{\Gamma} := \alpha(\Gamma_1)$. Dann erzeugt, nach Lemma 4.2, $\overline{\Gamma}$ den zu C_{Γ_2} linear

isometrischen Code $\overline{C}$. Nach Satz 4.1 gilt damit $\overline{\Gamma} \sim_E \Gamma_2$. Lemma 4.3 liefert aber auch $\Gamma_1 \sim_E \overline{\Gamma}$, womit aus der Transitivität von $\sim_E$ die Behauptung $\Gamma_1 \sim_E \Gamma_2$ folgt.

□

Aus Satz 4.2 folgt, dass Generatormatrizen zu semilinear isometrischen Codes im Sinne unserer Algorithmen auf jeden Fall als gleichwertig einzustufen sind. Eine exakte Erkennung von semilinearer Isometrie ist aber normalerweise sehr rechenaufwändig und würde den Programmablauf vermutlich derart stark bremsen, dass der positive Effekt (es werden niemals mehrere Vertreter einer semilinearen Isometrieklasse durchlaufen, aber auch niemals zwei Vertreter verschiedener Klassen als gleich eingestuft) diesen Nachteil nicht ausgleichen kann. Deshalb stellt sich die Frage, ob man nicht ein mit weniger Aufwand verbundenes Kriterium finden kann, das trotzdem „einigermaßen“ zuverlässig semilineare Isometrien erkennen kann. Grundsätzlich gibt es drei Typen solch „nichtexakter“ Kriterien: Zum einen diejenigen, bei denen gelegentlich Generatormatrizen nichtisometrischer Codes zu Unrecht miteinander identifiziert werden. Zum anderen die Kriterien, welche hin und wieder eine Isometrie „übersehen“. Und zuletzt solche, bei denen beides passieren kann. Für den heuristischen Ansatz erscheint ein Kriterium vom ersten Typ am zweckmäßigsten: Immerhin garantiert es, dass niemals an und für sich gleiche Matrizen erneut betrachtet werden; praktisch die gesamte Rechenzeit wird also zur Untersuchung von bisher noch nicht Aufgetretenem verwendet. Ein solches Kriterium soll nun abschließend vorgestellt werden:

4.5 Ein notwendiges Kriterium für semilineare Isometrie

Zu einem linearen (m, k, d, q)-Code C sei, für $1 \leq w, j \leq m$:

$$t_w^j(C) := |\{c = (c_1, c_2, ..., c_m) \in C : gew(c) = w,\ c_j \neq 0\}| \,,$$

d. h. $t_w^j(C)$ bezeichnet die Zahl der Codeworte von Gewicht w in C, deren j-te Komponente ungleich Null ist. Entsprechend sei dann $T^j(C) \in \mathbb{N}^m$ definiert durch

$$T^j(C) := \begin{pmatrix} t_1^j(C) \\ t_2^j(C) \\ \vdots \\ t_m^j(C) \end{pmatrix} .$$

Ist f_h nun eine Hashfunktion des Typs:

$$f_h : \mathbb{N}^m \to \{0, ..., 2^z - 1\} \,,$$

mit $z \in \mathbb{N}$ „ausreichend groß“, so können wir C wie folgt einen Hashwert zuweisen:

Definition 4.3 *Für einen linearen (m, k, d, q)-Code C sei*

$$hash(C) := \sum_{j=1...m} f_h(T^j(C)) \; (mod\, 2^z)\,.$$

Nun wollen wir beweisen, dass für zwei lineare Codes C_1 und C_2 die Gleichheit der Hashwerte $hash(C_1) = hash(C_2)$ eine notwendige Voraussetzung für das Vorliegen einer semilinearen Isometrie zwischen C_1 und C_2 darstellt:

Satz 4.3 *Sind C_1 und C_2 semilinear isometrische (m, k, d, q)-Codes, so gilt:*

$$hash(C_1) = hash(C_2)\,.$$

Beweis:
Da C_1 semilinear isometrisch zu C_2, gibt es eine semilineare Isometrie $\sigma = (\alpha, \iota)$, mit einer linearen Isometrie $\iota = \mu_\lambda \circ \pi$ (vgl. Abschnitt 1.6) und $\alpha \in Aut(\mathbb{F}_q)$, s. d. gilt: $C_2 = \iota(\alpha(C_1)) = \mu_\lambda \circ \pi \circ \alpha(C_1)$. Sei $C' := \alpha(C_1)$. Da α genau die Nichtnullkomponenten eines Codeworts aus C_1 wieder in Nichtnullkomponenten überführt, gilt offensichtlich

$$hash(C_1) = hash(C')\,.$$

Setze nun $C'' := \pi(C')$. π permutiert lediglich die Spalten von C', was nur eine Permutation der T^j zur Folge hat:

$$T^j(C'') = T^{\pi^{-1}(j)}(C')\,.$$

Aufgrund der Kommutativität der Summation ändert sich dadurch der Hashwert nach Definition 4.3 nicht, d. h. es folgt

$$hash(C') = hash(C'')\,.$$

Schließlich gilt, da μ_λ wiederum genau Nichtnullkomponten in ebensolche überführt:

$$hash(C'') = hash(\mu_\lambda(C'')) = hash(C_2)\,,$$

was die Gleichheitskette

$$hash(C_1) = hash(C') = hash(C'') = hash(C_2)$$

und damit die Behauptung ergibt.

□

Ein Beispiel, bei dem dieses Kriterium fälschlicherweise zwei nicht semilinear isometrische Codes als solche identifiziert, findet sich in Anhang B.

Kapitel 5

Zusammenfassung und Ausblick

Heuristische Verfahren kommen vor allem dort zum Einsatz, wo ein vollständiges Absuchen des Problemraumes unmöglich ist und sich das Treffen allgemeingültiger Aussagen als schwierig erweist. Natürlich muss aber auch dabei das behandelte Problem eine gewisse Struktur aufweisen: Es muss Zusammenhänge geben, die zwar nicht immer, aber zumindest „häufig“ gelten - ansonsten fehlen Ansatzpunkte für die Formulierung von Faustregeln. In unserem Fall ließ sich, für die Bewertungsfunktion *eval* aus Abschnitt 3.2, ausnutzen, dass für jedes $v \in \mathbb{F}_q^{k^*}$ die Dimension und damit die Ordnung des Orthogonalraums $v^\perp$ gleich ist. Dies ermöglichte die Berechnung eines *Schätzwertes* für die Wahrscheinlichkeit, dass die aktuelle Matrix nach *zufälliger* Hinzunahme der noch ausstehenden Zahl an Spalten einen Code mit (mindestens) der vorgeschriebenen Minimaldistanz generiert. Die Heuristik liegt in der Annahme, dass dieser Wert für Matrizen, bei denen eine solche Erweiterung tatsächlich existiert, in der Regel höher ausfällt als bei solchen, wo sie nicht möglich ist.

Obwohl diese Konstruktionsmethode für lineare Codes noch in den Kinderschuhen steckt, können sich die Ergebnisse (vgl. Anhang C) durchaus schon sehen lassen. Verbesserungspotential ist reichlich vorhanden, hier nur einige Möglichkeiten:

- Parallelisierung des Programmablaufs
- Bessere Kriterien für die *eval*-Funktion
- Variable Suchtiefe und variabler Verzweigungsfaktor im Suchbaum, gesteuert durch Heuristiken
- Genauere Isometrieerkennung
- Ausgefeilteres Backtrackingverhalten

Eine Anpassung für die Konstruktion speziellerer linearer Codes, etwa solche mit vorgeschriebener Automorphismengruppe, dürfte in den meisten Fällen problemlos realisierbar sein. Darüber hinaus ist auch eine Übertragung des Ansatzes auf verwandte Themengebiete, wie z. B. die Konstruktion von Designs, denkbar.

Anhang A

Exkurs: Ein Irrweg mit Erkenntnisgewinn

Die Idee zu dieser Arbeit kam mir während der Kombinatorik-Tagung ALCOMA im April 2005. Diese fand in Thurnau nahe Bayreuth statt, weshalb auch ich als Student mittleren Semesters die Gelegenheit zur Teilnahme nutzte. Naturgemäß gab es dort einige Vorträge, denen ich ab einem gewissen Zeitpunkt nicht mehr folgen konnte - dann setzte ich mich jeweils vor mein Notebook und begann, den gegen Ende von 2.2 vorgestellten Algorithmus weiter zu implementieren. Ich hielt es für erstrebenswert, dass die Verteilung der Nichtnullkomponenten auf die einzelnen Codeworte in jedem Schritt möglichst gleichmäßig bleibt. Der Einfachkeit halber beschränkte ich mich zunächst auf projektive, binäre Codes. Nach Lemma 2.3 gilt dann für die Gewichtssumme der Codevektoren, welche durch die im m-ten Schritt entstehende Matrix Γ_m erzeugt werden, stets:

$$\sum_{v \in \mathbb{F}_q^k} gew(v\Gamma_m) = m2^{k-1}\,.$$

Bei vollkommen gleichmäßiger Verteilung *hätten* alle von Null verschiedenen Vektoren als Gewicht das arithmetische Mittel

$$\rho_m = \frac{m2^{k-1}}{2^k - 1}\,,$$

wobei ρ_m aber in der Regel keine natürliche Zahl und die Idealverteilung somit nicht erreichbar ist. Es erschien mir zweckmäßig, für den Vergleichsoperator „$>$“ zunächst eine Bewertungsfunktion „*eval*“ wie folgt zu definieren (nämlich als das Quadrat des euklidischen Abstands zwischen dem Vektor, bei welchem das Gewicht eines jeden Codewortes ungleich Null eine Koordinate bildet und dem, der aus $2^k - 1$ Koordinaten mit dem Wert ρ_m besteht):

$$\Gamma \in \mathbb{F}_q^{k \times m} \Rightarrow eval(\Gamma) := \sum_{v \in \mathbb{F}_q^{k^*}} \left(gew(v\Gamma) - \rho_m\right)^2 .$$

$>$ setzte ich dann einfach so:

$$\Gamma_1 > \Gamma_2 :\Leftrightarrow eval(\Gamma_1) > eval(\Gamma_2)\,.$$

Das Ergebnis war höchst überraschend: Der Algorithmus wählte in jedem Schritt stets die erste untersuchte Spalte, denn allen Spalten (genauer: den jeweils durch ihre Hinzunahme entstehenden Matrizen) wurde durch *eval* der gleiche Wert zugeordnet. Dementsprechend schlecht waren die Resultate. Nachdem ich auf die Schnelle keinen Programmierfehler entdecken konnte, rechnete ich ein paar kleinere Beispiele von Hand nach. Und es stimmte tatsächlich: Der Wert von *eval* war für jede Matrix, die durch Hinzufügen einer (noch nicht vorhandenen) Nichtnullspalte zur bereits konstruierten Matrix Γ_{m-1} entstand, gleich. Aber es galt sogar noch mehr: Er hing gar nicht von Γ_{m-1} ab, sondern lediglich von m! Neugierig geworden, begann ich ein bißchen herumzurechnen:

Angenommen, es gilt für jeden projektiven, binären Code C der Blocklänge m:

$$\sum_{c\in C^*} (gew(c) - \rho_m)^2 = const(m,k)\,,$$

so ist dies äquivalent zu

$$\sum_{c\in C^*} gew(c)^2 - 2\rho_m \sum_{c\in C^*} gew(c) + \sum_{c\in C^*} \rho_m^2 = const(m,k)$$

und dies wiederum zu

$$\sum_{c\in C^*} gew(c)^2 = 2\rho_m \sum_{c\in C^*} gew(c) - (2^k-1)\rho_m^2 + const(m,k)\,.$$

Aber weil ja

$$\sum_{c\in C^*} gew(c) = m2^{k-1}\,,$$

folgt nach Ersetzen von ρ_m:

$$\sum_{c\in C^*} gew(c)^2 = \frac{m^2 2^{2k-1}}{2^k-1} - \frac{m^2 2^{2k-2}}{2^k-1} + const(m,k) = \frac{m^2 2^{2k-2}}{2^k-1} + const(m,k)\,.$$

Die rechte Seite hängt dabei nur noch von m und k ab - also hieße das, die Summe der Gewichtsquadrate eines projektiven, binären Codes ist stets eine nur von der Dimension und Blocklänge abhängige Konstante. Ich fragte einige der Teilnehmer der Konferenz, ob Ihnen ein derartiger Zusammenhang bekannt sei, aber niemand hatte bisher davon gehört. Einer meinte, wenn es wirklich zuträfe, so könnte es mit den sogenannten MacWilliams-Gleichungen zu tun haben. Und tatsächlich, ein Blick in [HP03, S. 259] brachte die gewünschte Bestätigung: Dort findet sich die letztlich aus der MacWilliams-Identität abgeleitete Gleichung:

$$\sum_{j=0}^{n} j^2 A_j = q^{k-2}[(q-1)n(qn-n+1) - (2qn-q-2n+2)A_1^\perp + 2A_2^\perp]\,.$$

q ist dabei die Ordnung des Grundkörpers, n die Blocklänge des untersuchten Codes C, A_j bezeichnet die Anzahl der Codeworte in C von Gewicht j und $A_j^\perp$ die entsprechende Anzahl für den zu C dualen Code $C^\perp$. Da in diesem Fall C projektiv ist, sind je zwei Spalten in der Generatormatrix linear unabhängig und darum (denn die Generatormatrix von C ist Kontrollmatrix von $C^\perp$) gibt es in $C^\perp$ keine Codeworte von Gewicht 1 bzw. 2, also $A_1^\perp = A_2^\perp = 0$. Einsetzen von $q = 2$ in die Gleichung liefert dann:

$$\left(\sum_{c \in C} gew(c)^2\right) = \sum_{j=0}^{n} j^2 A_j = 2^{k-2} n(n+1)\,.$$

Also gilt wirklich:

Lemma A.1 *Für einen binären, projektiven, linearen Code der Dimension k und der Blocklänge n ist*

$$\sum_{c \in C} gew(c)^2 = 2^{k-2} n(n+1)\,.$$

□

Etwas später stellte ich fest, dass man dieses Resultat auch durch eine relativ elementare Überlegung herleiten kann. Zur einfacheren Schreibweise wollen wir dazu noch folgendes einführen: Für $u, v \in \mathbb{F}_2^k$ sei

$$\langle u, v \rangle_{\mathbb{N}} := \begin{cases} 1_{\mathbb{N}} & falls\, \langle u, v \rangle = 1_{\mathbb{F}_2} \\ 0_{\mathbb{N}} & falls\, \langle u, v \rangle = 0_{\mathbb{F}_2} \end{cases}$$

Dabei ist $\langle u, v \rangle$ das gewöhnliche Standardskalarprodukt in $\mathbb{F}_2^k$. Nun zur alternativen Herleitung von Lemma A.1; es folgt direkt aus dem etwas allgemeineren

Lemma A.2 *Sind $v_1, v_2, ..., v_n \in \mathbb{F}_2^k$ paarweise verschiedene binäre Vektoren, so gilt:*

$$\sum_{u \in \mathbb{F}_2^k} (\sum_{i=1}^{n} \langle u, v_i \rangle_{\mathbb{N}})^2 = 2^{k-2} n(n+1)$$

Beweis (Induktion nach n):
$\underline{n = 1}$:

$$\sum_{u \in \mathbb{F}_2^k} \langle u, v_1 \rangle_{\mathbb{N}}^2 = |\mathbb{F}_2^k| - |\langle v_1 \rangle^\perp| = 2^k - 2^{k-1} = 2^{k-1}\,.$$

$\underline{(n-1) \to n}$:

$$\sum_{u \in \mathbb{F}_2^k} (\sum_{i=1}^{n} \langle u, v_i \rangle_{\mathbb{N}})^2 = \sum_{u \in \mathbb{F}_2^k} [(\sum_{i=1}^{n-1} \langle u, v_i \rangle_{\mathbb{N}})^2 + \langle u, v_n \rangle_{\mathbb{N}}^2 + 2\langle u, v_n \rangle_{\mathbb{N}} \sum_{i=1}^{n-1} \langle u, v_i \rangle_{\mathbb{N}}] =$$

$$\overset{IV}{=} 2^{k-2}(n-1)n + \sum_{u \in \mathbb{F}_2^k} \langle u, v_n \rangle_{\mathbb{N}}^2 + 2 \sum_{u \in \mathbb{F}_2^k} (\langle u, v_n \rangle_{\mathbb{N}} \sum_{i=1}^{n-1} \langle u, v_i \rangle_{\mathbb{N}}) =$$

$$= 2^{k-2}(n-1)n + 2^{k-1} + 2 \sum_{u \in \mathbb{F}_2^k \setminus \langle v_n \rangle^\perp} (\sum_{i=1}^{n-1} \langle u, v_i \rangle_{\mathbb{N}}) =$$

$$= 2^{k-2}(n^2 - n + 2) + 2 \sum_{i=1}^{n-1} (\sum_{u \in \mathbb{F}_2^k \setminus \langle v_n \rangle^\perp} \langle u, v_i \rangle_{\mathbb{N}}) =$$

$$= 2^{k-2}(n^2 - n + 2) + 2 \sum_{i=1}^{n-1} |\mathbb{F}_2^k \setminus (\langle v_n \rangle^\perp \cup \langle v_i \rangle^\perp)| =$$

$$\overset{(*)}{=} 2^{k-2}(n^2 - n + 2) + 2(n-1)2^{k-2} = 2^{k-2}n(n+1)\,,$$

$(*)\, wegen: \quad |\langle v_n \rangle^\perp \cup \langle v_i \rangle^\perp| = 3 \cdot 2^{k-2} \, \forall i \in \{1, ..., n-1\}\,.$

□

Anhang B

„Gegenbeispiel“ zum Isometriekriterium nach 4.5

Hier liefern wir, für die Parameter $n = 8$, $k = 3$, $q = 9$, ein Beispiel, in dem das notwendige Isometriekriterium aus 4.5 von zwei nicht semilinear isometrischen Codes erfüllt wird. Dies zeigt, dass es keinesfalls hinreichend für das Vorliegen einer semilinearen Isometrie ist:

Wir identifizieren zunächst $\mathbb{F}_9$ mit $\mathbb{F}_3[x]/(p)$, mit dem über $\mathbb{F}_3$ irreduziblen Polynom $p := 2x^2 + 2x + 1$. Ein primitives Element ist $\xi := x + (p)$. Für $i = 1..8$ wollen wir dann anstelle der Potenz $\xi^i \in \mathbb{F}_3[x]/(p)$ abkürzend die Ziffer i schreiben, die Ziffer 0 bezeichne das Nullelement. Damit ergeben sich folgende Verknüpfungstafeln:

+	0	8	4	1	7	6	5	2	3
0	0	8	4	1	7	6	5	2	3
8	8	4	0	7	6	1	2	3	5
4	4	0	8	6	1	7	3	5	2
1	1	7	6	5	2	3	0	8	4
7	7	6	1	2	3	5	8	4	0
6	6	1	7	3	5	2	4	0	8
5	5	2	3	0	8	4	1	7	6
2	2	3	5	8	4	0	7	6	1
3	3	5	2	4	0	8	6	1	7

bzw.

·	0	8	4	1	7	6	5	2	3
0	0	0	0	0	0	0	0	0	0
8	0	8	4	1	7	6	5	2	3
4	0	4	8	5	3	2	1	6	7
1	0	1	5	2	8	7	6	3	4
7	0	7	3	8	6	5	4	1	2
6	0	6	2	7	5	4	3	8	1
5	0	5	1	6	4	3	2	7	8
2	0	2	6	3	1	8	7	4	5
3	0	3	7	4	2	1	8	5	6

Betrachtet werden die Codes C_1 und C_2 zu den Generatormatrizen Γ_1 und Γ_2 mit

$$\Gamma_1 := \begin{pmatrix} 8 & 0 & 0 & 8 & 8 & 8 & 8 & 8 \\ 0 & 8 & 0 & 8 & 4 & 1 & 7 & 5 \\ 0 & 0 & 8 & 8 & 1 & 4 & 3 & 6 \end{pmatrix}, \; \Gamma_2 := \begin{pmatrix} 8 & 0 & 0 & 8 & 8 & 8 & 8 & 8 \\ 0 & 8 & 0 & 8 & 4 & 1 & 7 & 6 \\ 0 & 0 & 8 & 8 & 1 & 5 & 2 & 7 \end{pmatrix}.$$

Zum Vergleich führen wir, für die normierten Vertreter $\overline{v}$ jedes $\langle v \rangle \in PG_2(9)$, die zugehörigen Codeworte $\overline{v}\Gamma_1$ bzw. $\overline{v}\Gamma_2$ auf; alle anderen Worte ungleich Null entstehen daraus durch Multiplikation mit einem Element aus $\mathbb{F}_9^*$.

Repräsentanten aus C_1					
$\mathbf{gew(\overline{v}\Gamma_1) = 6}$		$\mathbf{gew(\overline{v}\Gamma_1) = 7}$		$\mathbf{gew(\overline{v}\Gamma_1) = 8}$	
$\overline{\mathbf{v}}$	$\overline{\mathbf{v}}\mathbf{\Gamma_1}$	$\overline{\mathbf{v}}$	$\overline{\mathbf{v}}\mathbf{\Gamma_1}$	$\overline{\mathbf{v}}$	$\overline{\mathbf{v}}\mathbf{\Gamma_1}$
008	**00881436**	**082**	**08232382**	**811**	**81126682**
080	**08084175**	**085**	**08527566**	**812**	**81241837**
081	**08175018**	**086**	**08611823**	**814**	**81417518**
083	**08358250**	**802**	**80235124**	**816**	**81654166**
084	**08403737**	**805**	**80521745**	**818**	**81868223**
087	**08760441**	**807**	**80764532**	**821**	**82148115**
088	**08846604**	**810**	**81072341**	**822**	**82214481**
800	**80088888**	**817**	**81733750**	**823**	**82376853**
801	**80173206**	**820**	**82031576**	**826**	**82682727**
803	**80350617**	**825**	**82563068**	**833**	**83367786**
804	**80402463**	**827**	**82707644**	**835**	**83512152**
806	**80616370**	**828**	**82855302**	**836**	**83645248**
808	**80847051**	**832**	**83278660**	**837**	**83781313**
813	**81305075**	**834**	**83434805**	**838**	**83823477**
815	**81580404**	**842**	**84222016**	**841**	**84115733**
824	**82420230**	**845**	**84557820**	**843**	**84338442**
830	**83056034**	**846**	**84661601**	**844**	**84443384**
831	**83100521**	**847**	**84770178**	**848**	**84886565**
840	**84004257**	**852**	**85260355**	**853**	**85311264**
850	**85027103**	**856**	**85603812**	**854**	**85458771**
851	**85184040**	**860**	**86013625**	**855**	**85575587**
863	**86342500**	**865**	**86508336**	**857**	**85746426**
866	**86630083**	**874**	**87471022**	**858**	**85832638**
870	**87065410**	**878**	**87810846**	**861**	**86151458**
872	**87206708**			**862**	**86287272**
880	**88040762**			**864**	**86466147**
887	**88718007**			**867**	**86725861**
888	**88801180**			**868**	**86874714**
				871	**87137367**
				873	**87383131**
				875	**87544673**
				876	**87628554**
				877	**87752285**
				881	**88162874**
				882	**88253543**
				883	**88324328**
				884	**88485656**
				885	**88536211**
				886	**88677435**

Repräsentanten aus C_2					
$\mathrm{gew}(\overline{v}\Gamma_2) = 6$		$\mathrm{gew}(\overline{v}\Gamma_2) = 7$		$\mathrm{gew}(\overline{v}\Gamma_2) = 8$	
$\overline{v}$	$\overline{v}\Gamma_2$	$\overline{v}$	$\overline{v}\Gamma_2$	$\overline{v}$	$\overline{v}\Gamma_2$
008	00881527	082	08232213	811	81126821
080	08084176	085	08527837	814	81417478
081	08175301	086	08611464	817	81733262
083	08358780	801	80173154	818	81868655
084	08403558	806	80616542	822	82214816
087	08760622	808	80847236	823	82376285
088	08846045	810	81072346	825	82563738
800	80088888	812	81241083	826	82682663
802	80235607	815	81580117	828	82855141
803	80350423	816	81654704	832	83278722
804	80402715	820	82031574	833	83367864
805	80521360	827	82707327	836	83645358
807	80764071	834	83434180	837	83781445
813	81305530	835	83512201	838	83823513
821	82148400	841	84115065	842	84222434
824	82420052	848	84886770	843	84338311
830	83056037	856	85603481	844	84443847
831	83100676	858	85832028	845	84557682
840	84004253	864	86466303	846	84661126
847	84770508	866	86630835	851	85184332
850	85027105	870	87065412	853	85311757
852	85260240	872	87206168	854	85458566
860	86013620	877	87752850	855	85575873
865	86508044	878	87810384	857	85746614
873	87383006			861	86151218
880	88040761			862	86287551
886	88677010			863	86342172
888	88801802			867	86725786
				868	86874467
				871	87137743
				874	87471631
				875	87544525
				876	87628277
				881	88162587
				882	88253375
				883	88324648
				884	88485224
				885	88536456
				887	88718133

Man erkennt durch einfaches Abzählen:

- $\forall j = 1, ..., 8 : t_6^j(C_1) = t_6^j(C_2) = 21 \cdot |\{\mathbb{F}_9^*\}| = 168$
- $\forall j = 1, ..., 8 : t_7^j(C_1) = t_7^j(C_2) = 21 \cdot |\{\mathbb{F}_9^*\}| = 168$
- $\forall j = 1, ..., 8 : t_8^j(C_1) = t_8^j(C_2) = 39 \cdot |\{\mathbb{F}_9^*\}| = 312$
- $\forall i : 1 \leq i \leq 5, \forall j = 1, ..., 8 : t_i^j(C_1) = t_i^j(C_2) = 0$

Daraus folgt unmittelbar:

$$\forall j = 1, ..., 8 : T^j(C_1) = T^j(C_2)\,,$$

und somit:

$$hash(C_1) = hash(C_2).$$

Aber Γ_1 und Γ_2 sind nicht erweiterungsäquivalent: Erweitert man Γ_2 um die Spalte

$$\gamma := \begin{pmatrix} 8 \\ 5 \\ 3 \end{pmatrix},$$

so generiert die entstehende Matrix $\Gamma_2' := (\Gamma_2 \,|\, \gamma)$ einen $(9, 3, 7, 9)$-Code mit Gewichtsverteilung $[7 - 9]\,(288, 144, 296)$. Dagegen ist es unmöglich, durch Erweiterung von Γ_1 um eine Spalte die Minimaldistanz auf sieben zu erhöhen:

Nehmen wir dazu an, es gäbe eine Spalte

$$\mu := \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{pmatrix}$$

s. d. $dist((\Gamma_1 \,|\, \mu)) = 7$. Sei $V_6 := \{v \in \mathbb{F}_9^3 : gew(v\Gamma_1) = 6\}$. Da $(800) \in V_6$, muss $\mu_1 \neq 0$ gelten. Weil wir außerdem ohne Einschränkung annehmen können, dass μ normiert ist, können wir $\mu_1 = 8$ setzen. Aus $\{(080), (830), (840), (850), (870), (880)\} \subset V_6$ folgt dann für die zweite Komponente μ_2:

- $8 \cdot \mu_2 \neq 0 \Leftrightarrow \mu_2 \neq 0$
- $8 + 3 \cdot \mu_2 \neq 0 \Leftrightarrow \mu_2 \neq 1$
- $8 + 4 \cdot \mu_2 \neq 0 \Leftrightarrow \mu_2 \neq 8$
- $8 + 5 \cdot \mu_2 \neq 0 \Leftrightarrow \mu_2 \neq 7$
- $8 + 7 \cdot \mu_2 \neq 0 \Leftrightarrow \mu_2 \neq 5$
- $8 + 8 \cdot \mu_2 \neq 0 \Leftrightarrow \mu_2 \neq 4$

Positiv formuliert ergibt das: $\mu_2 \in \{2, 3, 6\}$.

Da $\{(008), (801), (803), (804), (806), (808) \subset V_6$, kann man analog für μ_3 schließen:

- $8 \cdot \mu_3 \neq 0 \Leftrightarrow \mu_3 \neq 0$
- $8 + 1 \cdot \mu_3 \neq 0 \Leftrightarrow \mu_3 \neq 3$
- $8 + 3 \cdot \mu_3 \neq 0 \Leftrightarrow \mu_3 \neq 1$
- $8 + 4 \cdot \mu_3 \neq 0 \Leftrightarrow \mu_3 \neq 8$
- $8 + 6 \cdot \mu_3 \neq 0 \Leftrightarrow \mu_3 \neq 6$
- $8 + 8 \cdot \mu_3 \neq 0 \Leftrightarrow \mu_3 \neq 4$

Also folgt $\mu_3 \in \{2, 5, 7\}$.

Für jede der denkbaren Kombinationen $(\mu_2, \mu_3) \in \{2, 3, 6\} \times \{2, 5, 7\}$ findet sich aber mindestens ein $v \in V_6$, s. d. $\langle v, \mu \rangle = 0$:

- $(\mu_2, \mu_3) = (2, 2) \Rightarrow \langle (084), \mu \rangle = 0$
- $(\mu_2, \mu_3) = (2, 5) \Rightarrow \langle (081), \mu \rangle = 0$
- $(\mu_2, \mu_3) = (2, 7) \Rightarrow \langle (087), \mu \rangle = 0$
- $(\mu_2, \mu_3) = (3, 2) \Rightarrow \langle (824), \mu \rangle = 0$
- $(\mu_2, \mu_3) = (3, 5) \Rightarrow \langle (866), \mu \rangle = 0$
- $(\mu_2, \mu_3) = (3, 7) \Rightarrow \langle (088), \mu \rangle = 0$
- $(\mu_2, \mu_3) = (6, 2) \Rightarrow \langle (088), \mu \rangle = 0$
- $(\mu_2, \mu_3) = (6, 5) \Rightarrow \langle (815), \mu \rangle = 0$
- $(\mu_2, \mu_3) = (6, 7) \Rightarrow \langle (083), \mu \rangle = 0$

Daher wäre auf jeden Fall $dist((\Gamma_1 \,|\, \mu)) = 6$, im Widerspruch zur Annahme. Also gilt:

$$\forall \mu \in \mathbb{F}_9^k : dist((\Gamma_1 \,|\, \mu)) = 6\,.$$

Somit sind Γ_1 und Γ_2 nicht erweiterungsäquivalent, weshalb, nach Satz 4.2, C_1 und C_2 nicht semilinear isometrisch sein können.

Anhang C

Ergebnisse

Hier führen wir die Stellen auf, an denen mit Hilfe von ***„Heurico“*** die unteren Schranken der gängigen internationalen Tabellen (A. Brouwer, M. Grassl) für die optimalen Minimaldistanzen linearer Codes auf den Wert lb_{neu} verbessert werden konnten. In der Zeile zu ub stehen die oberen Schranken aus den Tabellen von M. Grassl (vergleiche **www.codetables.de**). Die angegebene Auswahl von Generatormatrizen wurde mit ***„Heurico“*** konstruiert; es gilt, dass zu jedem aufgeführten Parametertupel eine der Matrizen, entweder unmittelbar oder nach Streichen der benötigten Zahl von Spalten (Auswahl beliebig), einen linearen Code mit diesen Parametern erzeugt. Lediglich für den Fall $q = 2$, $k = 10$, bei dem mit einer von A. Kohnert konstruierten Generatormatrix für einen linearen $(177, 10, 84, 2)$-Code gestartet wurde, sei aus Platzgründen auf die angefügte CD verwiesen, auf der sich im Verzeichnis „GMs“ für jeden Parametersatz eine Generatormatrix findet.

Verbesserungen für q=2, k=10:

n	180	181	183	184
lb_{neu}	85	86	87	88
ub	86	87	88	88

Verbesserung für q=5, k=6:

n	47
lb_{neu}	32
ub	34

Verbesserungen für q=5, k=7:

n	19	33	37	44	51	52
lb_{neu}	10	20	23	28	33	34
ub	10	22	25	30	36	37

Verbesserung für q=7, k=4:

n	77
lb_{neu}	63
ub	64

Verbesserungen für q=7, k=5:

n	54	55	56	59	60	61	62	66	67	68
lb_{neu}	41	42	43	45	46	47	48	51	52	53
ub	44	45	46	49	49	49	50	54	55	56

Verbesserungen für q=7, k=6:

n	59	60	61	62	66	67	69	70	71	72	73	77
lb_{neu}	43	44	45	46	49	50	51	52	53	54	55	58
ub	48	49	49	49	53	54	56	56	57	58	59	63

Verbesserung für q=9, k=5:

n	33
lb_{neu}	25
ub	27

Generatormatrix für q=5, k=6:

Generatormatrix eines $(47, 6, 32, 5)$-Codes:

$$\begin{pmatrix}
1 0 0 0 0 0 1 0 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 \\
0 1 0 0 0 0 1 1 0 1 4 1 4 0 3 0 1 4 2 2 1 2 2 3 0 4 4 1 4 2 2 1 3 2 1 0 1 1 4 3 2 3 1 2 2 1 1 \\
0 0 1 0 0 0 1 1 1 2 0 4 3 4 2 3 2 3 2 0 1 3 2 1 1 2 1 3 3 0 0 0 0 2 1 4 3 0 1 2 3 0 4 0 2 0 3 \\
0 0 0 1 0 0 1 2 2 3 2 0 0 1 4 1 1 3 2 1 3 2 3 2 3 0 3 0 4 3 4 0 3 2 3 2 3 2 1 1 0 1 3 4 1 0 0 \\
0 0 0 0 1 0 1 3 4 4 2 2 4 3 4 1 3 1 1 2 4 2 3 1 1 2 0 0 3 0 0 2 0 0 1 0 3 2 3 0 1 3 2 4 1 3 2 \\
0 0 0 0 0 1 1 4 2 2 4 2 1 1 2 4 3 0 4 3 0 1 3 0 1 3 4 3 3 1 4 4 3 0 2 2 4 2 1 2 3 4 0 0 2 3 2
\end{pmatrix}$$

Generatormatrizen für q=5, k=7:

Generatormatrix eines $(19, 7, 10, 5)$-Codes:

$$\begin{pmatrix}
1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 1 1 \\
0 1 0 0 0 0 0 1 0 0 0 1 4 1 2 3 1 4 2 \\
0 0 1 0 0 0 0 1 0 2 1 2 3 3 1 3 1 1 1 \\
0 0 0 1 0 0 0 1 1 0 4 2 2 4 0 1 4 4 1 \\
0 0 0 0 1 0 0 1 1 2 2 1 2 0 4 2 3 3 1 \\
0 0 0 0 0 1 0 1 1 3 4 4 0 3 4 4 2 1 3 \\
0 0 0 0 0 0 1 1 2 3 1 0 2 4 2 3 2 4 0
\end{pmatrix}$$

Generatormatrix eines $(33, 7, 20, 5)$-Codes:

$$\begin{pmatrix}
1 0 0 0 0 0 0 1 0 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 \\
0 1 0 0 0 0 0 1 1 0 2 1 1 1 4 2 3 4 2 2 1 1 2 2 1 1 0 3 3 0 3 0 0 \\
0 0 1 0 0 0 0 1 1 1 3 4 3 0 2 1 2 2 2 0 4 1 3 3 1 0 4 2 1 2 3 2 2 \\
0 0 0 1 0 0 0 1 2 2 3 0 0 1 3 2 1 0 2 4 3 4 4 3 0 1 1 3 1 0 3 1 3 \\
0 0 0 0 1 0 0 1 2 3 0 2 1 4 1 4 4 4 3 3 1 1 3 4 0 0 2 3 3 4 2 3 1 \\
0 0 0 0 0 1 0 1 3 2 1 2 0 2 4 3 3 2 1 0 0 4 1 0 1 3 2 1 2 3 2 1 0 \\
0 0 0 0 0 0 1 1 4 3 2 3 3 2 3 1 2 1 1 2 0 3 0 4 4 3 2 3 1 0 1 3 0
\end{pmatrix}$$

Generatormatrix eines $(37, 7, 23, 5)$-Codes:

$$\begin{pmatrix}
1 0 0 0 0 0 0 1 0 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 1 \\
0 1 0 0 0 0 0 1 1 0 2 1 1 1 4 2 3 4 2 2 1 1 2 2 1 1 0 1 1 3 4 4 3 0 4 0 3 \\
0 0 1 0 0 0 0 1 1 1 3 4 3 0 2 1 2 2 2 0 4 1 3 3 1 0 4 0 4 1 4 4 1 0 0 1 2 \\
0 0 0 1 0 0 0 1 2 2 3 0 0 1 3 2 1 0 2 4 3 4 4 3 0 1 1 1 3 0 1 3 2 1 3 4 0 \\
0 0 0 0 1 0 0 1 2 3 0 2 1 4 1 4 4 4 3 3 1 1 3 4 0 0 2 1 2 1 1 4 0 0 3 3 3 \\
0 0 0 0 0 1 0 1 3 2 1 2 0 2 4 3 3 2 1 0 0 4 1 0 1 3 2 3 4 2 2 1 0 1 1 0 2 \\
0 0 0 0 0 0 1 1 4 3 2 3 3 2 3 1 2 1 1 2 0 3 0 4 4 3 2 2 2 0 3 4 2 2 0 2 4
\end{pmatrix}$$

Generatormatrix eines $(45, 7, 29, 5)$-Codes:

$$\begin{pmatrix}
1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,0\,1\,1\,1\,0\,1\,1\,1\,1\,1\\
0\,1\,0\,0\,0\,0\,0\,1\,0\,1\,1\,1\,1\,0\,1\,4\,1\,3\,0\,2\,2\,1\,0\,4\,2\,2\,3\,1\,3\,3\,3\,4\,4\,0\,3\,1\,4\,0\,0\,0\,2\,4\,0\,4\,4\\
0\,0\,1\,0\,0\,0\,0\,1\,1\,0\,1\,3\,4\,0\,4\,3\,1\,0\,1\,3\,0\,1\,4\,2\,2\,0\,0\,2\,2\,4\,0\,4\,0\,4\,4\,4\,3\,0\,2\,1\,2\,2\,2\,2\,4\\
0\,0\,0\,1\,0\,0\,0\,1\,1\,2\,0\,4\,1\,1\,0\,2\,3\,4\,0\,2\,0\,1\,4\,2\,4\,4\,2\,4\,3\,1\,4\,1\,2\,2\,0\,0\,0\,2\,3\,4\,1\,0\,3\,2\,1\\
0\,0\,0\,0\,1\,0\,0\,1\,2\,1\,2\,4\,2\,4\,2\,0\,3\,3\,1\,1\,3\,0\,3\,4\,2\,1\,4\,1\,2\,2\,0\,4\,0\,4\,4\,0\,2\,2\,4\,2\,0\,3\,0\,1\,2\\
0\,0\,0\,0\,0\,1\,0\,1\,2\,4\,4\,1\,0\,3\,2\,3\,4\,1\,1\,1\,0\,3\,0\,0\,2\,0\,2\,1\,4\,4\,4\,2\,1\,4\,1\,1\,1\,0\,3\,1\,3\,0\,0\,4\,3\\
0\,0\,0\,0\,0\,0\,1\,1\,3\,4\,2\,2\,4\,3\,3\,2\,3\,3\,3\,0\,2\,1\,4\,0\,4\,0\,0\,2\,1\,3\,0\,3\,3\,4\,4\,1\,4\,4\,4\,4\,4\,4\,2\,0\,1
\end{pmatrix}$$

Generatormatrix eines $(52, 7, 34, 5)$-Codes:

$$\begin{pmatrix}
1\,0\,0\,0\,0\,0\,0\,1\,0\,1\,1\,0\,1\,1\,1\,1\,1\,1\,1\,0\,0\,1\,1\,1\,1\,0\,1\,1\,1\,1\,1\,1\,1\,0\,1\,0\,1\,1\,1\,1\,0\,0\,0\,1\,1\,1\,0\,1\,1\\
0\,1\,0\,0\,0\,0\,0\,1\,1\,0\,2\,1\,1\,1\,4\,2\,3\,4\,2\,2\,1\,1\,2\,2\,1\,1\,0\,1\,1\,3\,4\,4\,3\,1\,4\,1\,2\,1\,4\,1\,2\,1\,1\,1\,1\,0\,4\,2\,4\,1\,2\,0\\
0\,0\,1\,0\,0\,0\,0\,1\,1\,1\,3\,4\,3\,0\,2\,1\,2\,2\,2\,0\,4\,1\,3\,3\,1\,0\,4\,0\,4\,1\,4\,4\,1\,0\,3\,2\,0\,0\,0\,4\,3\,1\,4\,1\,2\,4\,4\,0\,4\,2\,4\,3\\
0\,0\,0\,1\,0\,0\,0\,1\,2\,2\,3\,0\,0\,1\,3\,2\,1\,0\,2\,4\,3\,4\,4\,3\,0\,1\,1\,1\,3\,0\,1\,3\,2\,0\,2\,3\,0\,2\,1\,4\,1\,2\,4\,0\,0\,3\,0\,3\,1\,2\,4\,4\\
0\,0\,0\,0\,1\,0\,0\,1\,2\,3\,0\,2\,1\,4\,1\,4\,4\,4\,3\,3\,1\,1\,3\,4\,0\,0\,2\,1\,2\,1\,1\,4\,0\,0\,0\,1\,1\,2\,4\,4\,1\,2\,1\,3\,2\,1\,1\,2\,4\,0\,0\,0\\
0\,0\,0\,0\,0\,1\,0\,1\,3\,2\,1\,2\,0\,2\,4\,3\,3\,2\,1\,0\,0\,4\,1\,0\,1\,3\,2\,3\,4\,2\,2\,1\,0\,2\,4\,4\,3\,0\,3\,0\,0\,3\,3\,3\,2\,0\,1\,2\,4\,0\,0\,2\\
0\,0\,0\,0\,0\,0\,1\,1\,4\,3\,2\,3\,3\,2\,3\,1\,2\,1\,1\,2\,0\,3\,0\,4\,4\,3\,2\,2\,2\,0\,3\,4\,2\,1\,0\,2\,4\,2\,1\,3\,3\,2\,3\,3\,1\,4\,2\,1\,1\,3\,3\,2
\end{pmatrix}$$

Generatormatrix für q=7, k=4:

Generatormatrix eines $(77, 4, 63, 7)$-Codes:

$$\begin{pmatrix}
1\,0\,0\,0\,1\,1\,0\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,0\,1\,0\,1\,1\,1\,1\,0\,1\,0\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,0\,1\,1\,1\,1\,1\,1\,1\,1\,0\,1\,1\,1\,1\,1\,1\,0\,1\,1\,0\\
0\,1\,0\,0\,1\,2\,1\,0\,3\,1\,2\,6\,4\,3\,4\,1\,6\,5\,5\,2\,0\,3\,0\,3\,6\,3\,5\,0\,1\,1\,1\,6\,0\,4\,6\,1\,1\,1\,4\,6\,1\,0\,2\,4\,2\,5\,6\,1\,1\,5\,2\,2\,3\,3\,1\,5\,2\,3\,5\,6\,1\,4\,0\,4\,1\,5\,3\,6\,0\,6\,0\,4\,0\,4\,3\,0\\
0\,0\,1\,0\,1\,3\,2\,1\,4\,0\,4\,6\,3\,2\,6\,3\,2\,5\,2\,1\,4\,6\,2\,5\,5\,4\,6\,1\,5\,4\,0\,5\,5\,3\,1\,0\,0\,4\,6\,0\,4\,6\,5\,3\,2\,2\,5\,0\,1\,4\,1\,4\,6\,3\,3\,2\,6\,0\,2\,6\,6\,3\,3\,0\,2\,1\,1\,0\,4\,3\,1\,5\,0\,1\,6\,2\,1\\
0\,0\,0\,1\,1\,4\,2\,4\,1\,5\,2\,0\,2\,4\,5\,3\,1\,2\,5\,3\,5\,2\,3\,4\,3\,0\,4\,6\,1\,3\,2\,4\,4\,0\,1\,0\,5\,0\,6\,4\,5\,5\,2\,3\,3\,2\,3\,1\,3\,2\,0\,1\,1\,5\,0\,6\,6\,4\,1\,0\,3\,6\,1\,2\,4\,5\,4\,0\,6\,4\,6\,0\,5\,1\,3\,5\,3
\end{pmatrix}$$

Generatormatrizen für q=7, k=5:

Generatormatrix eines $(56, 5, 43, 7)$-Codes:

$$\begin{pmatrix}
1\,0\,0\,0\,0\,1\,1\,0\,1\,1\,1\,1\,1\,1\,0\,1\,1\,1\,0\,1\,1\,1\,1\,1\,1\,0\,1\,1\,1\,1\,1\,1\,1\,0\,1\,1\,1\,0\,1\,1\,1\,1\,1\,1\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\\
0\,1\,0\,0\,0\,1\,2\,1\,0\,1\,2\,4\,6\,1\,1\,3\,2\,4\,6\,0\,5\,3\,3\,6\,1\,5\,5\,3\,1\,3\,1\,2\,0\,1\,3\,5\,2\,5\,0\,1\,1\,1\,5\,4\,6\,2\,6\,2\,0\,0\,4\,2\,3\,2\,5\,5\\
0\,0\,1\,0\,0\,1\,3\,1\,1\,0\,4\,6\,5\,6\,4\,2\,0\,2\,6\,1\,0\,1\,4\,2\,3\,5\,1\,2\,0\,0\,2\,5\,5\,3\,4\,2\,3\,4\,3\,5\,3\,0\,1\,1\,5\,2\,6\,0\,4\,4\,3\,5\,4\,3\,0\,6\\
0\,0\,0\,1\,0\,1\,4\,4\,2\,2\,1\,0\,0\,6\,1\,2\,1\,5\,1\,5\,2\,6\,5\,6\,6\,4\,6\,4\,2\,2\,1\,5\,3\,0\,6\,5\,5\,4\,1\,4\,4\,6\,0\,2\,5\,3\,6\,2\,4\,1\,4\,3\,6\,2\,1\,3\\
0\,0\,0\,0\,1\,1\,5\,3\,2\,3\,4\,1\,3\,4\,2\,4\,3\,0\,2\,1\,4\,0\,2\,2\,0\,1\,3\,2\,1\,5\,3\,2\,4\,4\,3\,3\,4\,5\,0\,4\,5\,5\,0\,3\,5\,5\,3\,6\,6\,5\,2\,6\,4\,5\,1\,4
\end{pmatrix}$$

Generatormatrix eines $(62, 5, 48, 7)$-Codes:

$$\begin{pmatrix}
1\,0\,0\,0\,0\,1\,1\,0\,1\,1\,1\,1\,1\,1\,0\,1\,1\,1\,0\,1\,1\,1\,1\,1\,1\,0\,1\,1\,1\,1\,1\,1\,1\,0\,1\,1\,1\,0\,1\,1\,1\,1\,1\,1\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\\
0\,1\,0\,0\,0\,1\,2\,1\,0\,1\,2\,4\,6\,1\,1\,3\,2\,4\,6\,0\,5\,3\,3\,6\,1\,5\,5\,3\,1\,3\,1\,2\,0\,1\,3\,5\,2\,5\,0\,1\,1\,1\,5\,4\,6\,2\,6\,2\,0\,0\,2\,4\,4\,0\,4\,2\,0\,3\,5\,1\,3\,2\\
0\,0\,1\,0\,0\,1\,3\,1\,1\,0\,4\,6\,5\,6\,4\,2\,0\,2\,6\,1\,0\,1\,4\,2\,3\,5\,1\,2\,0\,0\,2\,5\,5\,3\,4\,2\,3\,4\,3\,5\,3\,0\,1\,1\,5\,2\,6\,0\,4\,4\,5\,5\,3\,0\,4\,2\,0\,6\,1\,1\,6\,0\\
0\,0\,0\,1\,0\,1\,4\,4\,2\,2\,1\,0\,0\,6\,1\,2\,1\,5\,1\,5\,2\,6\,5\,6\,6\,4\,6\,4\,2\,2\,1\,5\,3\,0\,6\,5\,5\,4\,1\,4\,4\,6\,0\,2\,5\,3\,6\,2\,4\,1\,3\,4\,4\,6\,4\,4\,1\,3\,3\,5\,1\,5\\
0\,0\,0\,0\,1\,1\,5\,3\,2\,3\,4\,1\,3\,4\,2\,4\,3\,0\,2\,1\,4\,0\,2\,2\,0\,1\,3\,2\,1\,5\,3\,2\,4\,4\,3\,3\,4\,5\,0\,4\,5\,5\,0\,3\,5\,5\,3\,6\,6\,5\,6\,5\,2\,5\,3\,2\,1\,1\,2\,5\,6\,5
\end{pmatrix}$$

Generatormatrix eines $(68, 5, 53, 7)$-Codes:

$$\begin{pmatrix}
10000110111111011110111111110111101111100111111111111111111110111011\\
01000121012461132460533615531312013525011154626200244042030211512150\\
00100131104656420261014235120025534234353011526044553042044662051153\\
00010144221006121515265664642215306554144602536241344644132606650025\\
00001153234134243021402201321532443345045503553665652532144111243122
\end{pmatrix}$$

Generatormatrizen für q=7, k=6:

Generatormatrix eines $(62, 6, 46, 7)$-Codes:

$$\begin{pmatrix}
10000011011111111111011110110110111111011111110111111111111111\\
01000012136361520650152114514214341231234325013526156566613211\\
00100013164015643411034020632111142040523654203211545033004112\\
00010014220563626612333554440323250033120454041425630642022020\\
00001015215322031631504001465510066250234233133164601223044613\\
00000116461542200255042642545364034352534600116132326114635435
\end{pmatrix}$$

Generatormatrix eines $(67, 6, 50, 7)$-Codes:

$$\begin{pmatrix}
10000011011111111111011110110110111111011111110111111111111111011111111\\
01000012136361520650152114514214341231234325013526156566611101116041\\
00100013164015643411034020632111142040523654203211545033005166126360\\
00010014220563626612333554440323250033120454041425630642021225664520\\
00001015215322031631504001465510066250234233133164601223043256016160\\
00000116461542200255042642545364034352534600116132326114633614644055
\end{pmatrix}$$

Generatormatrix eines $(73, 6, 55, 7)$-Codes:

$$\begin{pmatrix}
10000011011111111111011110110110111111011111110111111111111111011111100110111\\
01000012136361520650152114514214341231234325013526156566611101116111214021\\
00100013164015643411034020632111142040523654203211545033005166126365022650\\
00010014220563626612333554440323250033120454041425630642021225664102452420\\
00001015215322031631504001465510066250234233133164601223043256016006531640\\
00000116461542200255042642545364034352534600116132326114633614644051103210
\end{pmatrix}$$

Generatormatrix eines $(77, 6, 58, 7)$-Codes:

$$\begin{pmatrix}
10000011011111111111011110110110111111011111110111111111111111011111100111101111\\
01000012136361520650152114514214341231234325013526156566611101116111201164656\\
00100013164015643411034020632111142040523654203211545033005166126365013234226\\
00010014220563626612333554440323250033120454041425630642021225664102466516421\\
00001015215322031631504001465510066250234233133164601223043256016006524361033\\
00000116461542200255042642545364034352534600116132326114633614644051155060324
\end{pmatrix}$$

Generatormatrix für q=9, k=5: (Verknüpfungen: siehe Anhang B)

Generatormatrix eines $(33, 5, 25, 9)$-Codes:

$$\begin{pmatrix}
8&0&0&0&0&8&8&8&8&0&8&8&8&8&8&8&8&8&8&8&8&8&8&8&8&8&0&8&8&8&0&8&8\\
0&8&0&0&0&8&4&1&6&8&8&2&4&3&5&5&6&2&0&2&1&8&4&5&2&0&8&5&3&4&8&5&2\\
0&0&8&0&0&8&1&5&7&8&5&6&7&4&4&6&0&4&5&4&3&5&0&1&7&3&3&7&2&7&8&5&6\\
0&0&0&8&0&8&7&2&3&2&7&0&2&1&8&4&7&4&3&1&1&2&2&4&0&5&0&1&3&6&3&8&8\\
0&0&0&0&8&8&6&7&5&1&4&6&2&8&5&0&1&2&1&1&2&5&4&7&8&5&5&6&1&7&8&6&0
\end{pmatrix}$$

Literaturverzeichnis

[BBF+06] A. Betten, M. Braun, H. Fripertinger, A. Kerber, A. Kohnert, and A. Wassermann. *Error-correcting linear codes. Classification by isometry and applications.* Algorithms and Computation in Mathematics. Springer Verlag, Berlin, 2006.

[Bet] A. Betten. *distance optimal linear codes over GF(2).* `http://www.math.colostate.edu/~betten/research/codes/GF2/codes_GF2.html`.

[HP03] W.C. Huffman and V. Pless. *Fundamentals of Error-Correcting Codes.* Cambridge University Press, Cambridge, 2003.

Erklärung

Hiermit erkläre ich, dass ich diese Diplomarbeit selbst verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe. Sie wurde noch in keinem anderen Prüfungsverfahren vorgelegt.

Bayreuth, im Februar 2007

(Johannes Zwanzger)